OpenShift Container Platform 4.16

インストール後の設定

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

Last Updated: 2024-07-02

OpenShift Container Platform 4.16 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

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

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
- クラスター
- ノード
- 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. インストール後の設定タスク

インストール後の設定タスクを実行して、ニーズに合わせて環境を設定できます。

以下のリストは、これらの設定の詳細です。

- 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、ノードとカスタムリソースを設定できます。
- ベアメタルノードの設定: Bare Metal Operator (BMO) を使用してベアメタルホストを管理できます。BMO は次の操作を完了できます。
  - ホストのハードウェアの詳細を検査し、ベアメタルホストに報告します。
  - ファームウェアを検査し、BIOS を設定します。
  - 必要なイメージでホストをプロビジョニングします。
  - ホストをプロビジョニングする前または後に、ホストのディスクの内容をクリーンアップします。
- クラスター機能の設定: OpenShift Container Platform クラスターの以下の機能を変更できます。
  - 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
  - アイデンティティープロバイダー
  - 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

- プライベートクラスターの設定: デフォルトでは、インストールプログラムはパブリックにアクセス可能な DNS と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。内部ネットワーク内からのみクラスターにアクセスできるようにするには、次のコンポーネントを設定してプライベートにします。
  - 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) コンピュートマシンを使用します。次のノード操作を実行できます。
  - コンピュートマシンの追加および削除
  - テイントおよび容認の削除
  - ノードあたりの Pod の最大数の設定
  - Device Manager の有効化
- ネットワークの設定: OpenShift Container Platform をインストールした後、以下のコンポーネントを設定できます。
  - Ingress クラスタートラフィック
  - 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ーの有効化
- ストレージの設定: デフォルトでは、コンテナー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。次のタスクを実行するように OAuth を設定できます。
- アイデンティティー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
- OperatorHub から Operator をインストールする

- アラート通知の設定: デフォルトでは、アラートの発生は 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 4.16 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ーを確保するには、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 installer-provisioned infrastructure 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。

Ingress Operator は、カスタムのデフォルト証明書を設定するまで、プレースホルダーとして機能する Ingress コントローラーのデフォルト証明書を生成します。実稼働クラスターで Operator が生成するデフォルト証明書は使用しないでください。Ingress Operator は、独自の署名証明書または生成するデフォルト証明書をローテーションしません。Operator が生成するデフォルト証明書は、設定するカスタムデフォルト証明書のプレースホルダーとして使用されます。

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

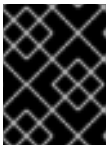
```
apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    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>-wfpg4: 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

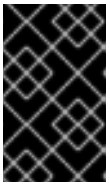
クラスターを Amazon Web Services (AWS) または Microsoft Azure にデプロイした後に、プライベートゾーンのみを使用するように API サーバーを再設定することができます。

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. クラウドプロバイダーの Web ポータルまたはコンソールで、次の操作を行います。
   a. 適切なロードバランサーコンポーネントを見つけて削除します。
      - 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 の場合は、パブリックロードバランサーの **api-internal-v4** ルールを削除します。
   b. Azure の場合、Ingress Controller エンドポイントの公開スコープを **Internal** に設定します。詳細は、「Ingress Controller エンドポイント公開スコープを内部に設定」を参照してください。
   c. Azure パブリックロードバランサーの場合は、Ingress Controller エンドポイントの公開スコープを **Internal** に設定し、パブリックロードバランサーに既存の受信規則がない場合は、バックエンドアドレスプールに送信トラフィックを提供するための送信規則を明示的に作成する必要があります。詳細は、送信ルールの追加に関する Microsoft Azure のドキュメントを参照してください。
   d. パブリックゾーンの **api.$clustername.$yourdomain** または **api.$clustername** DNS エントリーを削除します。
2. AWS クラスター: 外部ロードバランサーを削除します。



重要

以下の手順は、installer-provisioned infrastructure (IPI) のクラスターでのみ実行できます。user-provisioned infrastructure (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。

- クラスターがコントロールプレーンマシンセットを使用する場合は、コントロールプレーンマシンセットのカスタムリソースで、パブリックまたは外部ロードバランサーを設定する行を削除します。

```
# ...
providerSpec:
  value:
# ...
    loadBalancers:
    - name: lk4pj-ext (1)
```

```
    type: network (2)
  - name: lk4pj-int
    type: network
# ...
```

(1) **-ext** で終わる外部ロードバランサーの **name** 値を削除します。

(2) 外部ロードバランサーの **type** 値を削除します。

- クラスターがコントロールプレーンマシンセットを使用しない場合は、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する必要があります。

  i. ターミナルから、次のコマンドを実行して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
  ```

  コントロールプレーンマシンの名前には **master** が含まれています。

  ii. 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。

  A. 次のコマンドを実行して、コントロールプレーンマシンオブジェクトを編集します。

  ```
  $ oc edit machines -n openshift-machine-api <control_plane_name> (1)
  ```

  (1) 変更するコントロールプレーンマシンオブジェクトの名前を指定します。

  B. 次の例でマークされている、外部ロードバランサーを説明する行を削除します。

  ```
  # ...
  providerSpec:
    value:
  # ...
      loadBalancers:
      - name: lk4pj-ext (1)
        type: network (2)
      - name: lk4pj-int
        type: network
  # ...
  ```

  (1) **-ext** で終わる外部ロードバランサーの **name** 値を削除します。

  (2) 外部ロードバランサーの **type** 値を削除します。

C. 変更を保存して、オブジェクト仕様を終了します。

D. コントロールプレーンマシンごとに、このプロセスを繰り返します。

関連情報

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

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 がインストールされている。

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。

2.5. AZURE 上でプライベートストレージエンドポイントを設定する

Image Registry Operator を利用すると、Azure 上でプライベートエンドポイントを使用できます。これにより、OpenShift Container Platform がプライベート Azure クラスターにデプロイされている場合に、プライベートストレージアカウントのシームレスな設定が可能になります。これにより、パブリック向けのストレージエンドポイントを公開せずにイメージレジストリーをデプロイできます。

次のどちらかの方法で、Azure 上のプライベートストレージエンドポイントを使用するように Image Registry Operator を設定できます。

- Image Registry Operator を設定して VNet 名とサブネット名を検出する
- ユーザーが指定した Azure 仮想ネットワーク (VNet) 名とサブネット名を使用する

2.5.1. Azure 上でプライベートストレージエンドポイントを設定する場合の制限事項

Azure 上でプライベートストレージエンドポイントを設定する場合は、次の制限が適用されます。

- プライベートストレージエンドポイントを使用するように Image Registry Operator を設定すると、ストレージアカウントへのパブリックネットワークアクセスが無効になります。したがって、OpenShift Container Platform の外部のレジストリーからイメージをプルするには、レジストリーの Operator 設定で **disableRedirect: true** を設定する必要があります。リダイレクトが有効になっていると、ストレージアカウントからイメージを直接プルするように、レジストリーによってクライアントがリダイレクトされますが、これは機能しません。パブリックネットワークアクセスが無効になっているためです。詳細は、「Azure でプライベートストレージエンドポイントを使用する場合にリダイレクトを無効にする」を参照してください。

- この操作は、Image Registry Operator によって元に戻すことはできません。

2.5.2. Image Registry Operator による VNet 名とサブネット名の検出を有効にして Azure 上でプライベートストレージエンドポイントを設定する

次の手順では、VNet 名とサブネット名を検出するように Image Registry Operator を設定して、Azure 上でプライベートストレージエンドポイントを設定する方法を示します。

前提条件

- Azure 上で動作するようにイメージレジストリーを設定している。

- ネットワークが、Installer Provisioned Infrastructure インストール方法を使用してセットアップされている。
  カスタムネットワーク設定を使用するユーザーの場合は、「ユーザー指定の VNet 名とサブネット名を使用して Azure 上でプライベートストレージエンドポイントを設定する」を参照してください。

手順

1. Image Registry Operator の **config** オブジェクトを編集し、**networkAccess.type** を **Internal** に設定します。

```
$ oc edit configs.imageregistry/cluster
```

```
# ...
spec:
  # ...
   storage:
     azure:
       # ...
       networkAccess:
         type: Internal
# ...
```

2. オプション: 次のコマンドを入力して、Operator がプロビジョニングを完了したことを確認します。これには数分かかる場合があります。

```
$ oc get configs.imageregistry/cluster -o=jsonpath="
{.spec.storage.azure.privateEndpointName}" -w
```

3. オプション: レジストリーがルートによって公開されており、ストレージアカウントをプライベートに設定する場合、クラスターの外部へのプルを引き続き機能させるには、リダイレクトを無効にする必要があります。次のコマンドを入力して、Image Operator 設定のリダイレクトを無効にします。

```
$ oc patch configs.imageregistry cluster --type=merge -p '{"spec":{"disableRedirect": true}}'
```

注記

リダイレクトが有効になっていると、クラスターの外部からのイメージのプルが機能しなくなります。

検証

1. 次のコマンドを実行して、レジストリーサービス名を取得します。

```
$ oc registry info --internal=true
```

出力例

```
image-registry.openshift-image-registry.svc:5000
```

2. 次のコマンドを実行してデバッグモードに入ります。

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

3. 推奨される **chroot** コマンドを実行します。以下に例を示します。

```
$ chroot /host
```

4. 次のコマンドを入力して、コンテナーレジストリーにログインします。

```
$ podman login --tls-verify=false -u unused -p $(oc whoami -t) image-registry.openshift-image-registry.svc:5000
```

出力例

```
Login Succeeded!
```

5. 次のコマンドを入力して、レジストリーからイメージをプルできることを確認します。

```
$ podman pull --tls-verify=false image-registry.openshift-image-registry.svc:5000/openshift/tools
```

出力例

```
Trying to pull image-registry.openshift-image-registry.svc:5000/openshift/tools/openshift/tools...
Getting image source signatures
Copying blob 6b245f040973 done
Copying config 22667f5368 done
```

```
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

2.5.3. ユーザー指定の VNet 名とサブネット名を使用して Azure 上でプライベートストレージエンドポイントを設定する

次の手順を使用して、パブリックネットワークアクセスが無効な、Azure 上のプライベートストレージエンドポイントの背後で公開されるストレージアカウントを設定します。

前提条件

- Azure 上で動作するようにイメージレジストリーを設定している。
- Azure 環境で使用する VNet 名とサブネット名を把握している。
- ネットワークが Azure の別のリソースグループに設定されている場合は、そのリソースグループの名前も把握している。

手順

1. Image Registry Operator の **config** オブジェクトを編集し、VNet 名とサブネット名を使用してプライベートエンドポイントを設定します。

```
$ oc edit configs.imageregistry/cluster
```

```
# ...
spec:
  # ...
   storage:
     azure:
       # ...
       networkAccess:
         type: Internal
         internal:
           subnetName: <subnet_name>
           vnetName: <vnet_name>
           networkResourceGroupName: <network_resource_group_name>
# ...
```

2. オプション: 次のコマンドを入力して、Operator がプロビジョニングを完了したことを確認します。これには数分かかる場合があります。

```
$ oc get configs.imageregistry/cluster -o=jsonpath="
{.spec.storage.azure.privateEndpointName}" -w
```

注記

リダイレクトが有効になっていると、クラスターの外部からのイメージのプルが機能しなくなります。

検証

1. 次のコマンドを実行して、レジストリーサービス名を取得します。

```
$ oc registry info --internal=true
```

出力例

```
image-registry.openshift-image-registry.svc:5000
```

2. 次のコマンドを実行してデバッグモードに入ります。

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

3. 推奨される **chroot** コマンドを実行します。以下に例を示します。

```
$ chroot /host
```

4. 次のコマンドを入力して、コンテナーレジストリーにログインします。

```
$ podman login --tls-verify=false -u unused -p $(oc whoami -t) image-registry.openshift-image-registry.svc:5000
```

出力例

```
Login Succeeded!
```

5. 次のコマンドを入力して、レジストリーからイメージをプルできることを確認します。

```
$ podman pull --tls-verify=false image-registry.openshift-image-registry.svc:5000/openshift/tools
```

出力例

```
Trying to pull image-registry.openshift-image-registry.svc:5000/openshift/tools/openshift/tools...
Getting image source signatures
Copying blob 6b245f040973 done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

2.5.4. オプション: Azure でプライベートストレージエンドポイントを使用する場合にリダイレクトを無効にする

デフォルトでは、イメージレジストリーを使用する場合、リダイレクトが有効になります。リダイレクトにより、レジストリー Pod からオブジェクトストレージへのトラフィックのオフロードが可能になり、プルが高速化されます。リダイレクトが有効で、ストレージアカウントがプライベートである場合、クラスターの外部のユーザーはレジストリーからイメージをプルできません。

場合によっては、クラスターの外部のユーザーがレジストリーからイメージをプルできるように、リダイレクトを無効にする必要があります。

リダイレクトを無効にするには、次の手順を実行します。

前提条件

- Azure 上で動作するようにイメージレジストリーを設定している。
- ルートを設定している。

手順

- 次のコマンドを入力して、イメージレジストリー設定のリダイレクトを無効にします。

```
$ oc patch configs.imageregistry cluster --type=merge -p '{"spec":{"disableRedirect": true}}'
```

検証

1. 次のコマンドを実行して、レジストリーサービス名を取得します。

```
$ oc registry info
```

出力例

```
default-route-openshift-image-registry.<cluster_dns>
```

2. 次のコマンドを入力して、コンテナーレジストリーにログインします。

```
$ podman login --tls-verify=false -u unused -p $(oc whoami -t) default-route-openshift-image-registry.<cluster_dns>
```

出力例

```
Login Succeeded!
```

3. 次のコマンドを入力して、レジストリーからイメージをプルできることを確認します。

```
$ podman pull --tls-verify=false default-route-openshift-image-registry.<cluster_dns>/openshift/tools
```

出力例

```
Trying to pull default-route-openshift-image-registry.<cluster_dns>/openshift/tools...
Getting image source signatures
Copying blob 6b245f040973 done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

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 はこれらのタスクを完了するために次のリソースを使用します。

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

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、**HostFirmwareSettings** リソース、**FirmwareSchema** リソース、および **HostFirmwareComponents** リソースを使用して、ファームウェア仕様の詳細を指定し、ベアメタルホストのファームウェアをアップグレードまたはダウングレードします。

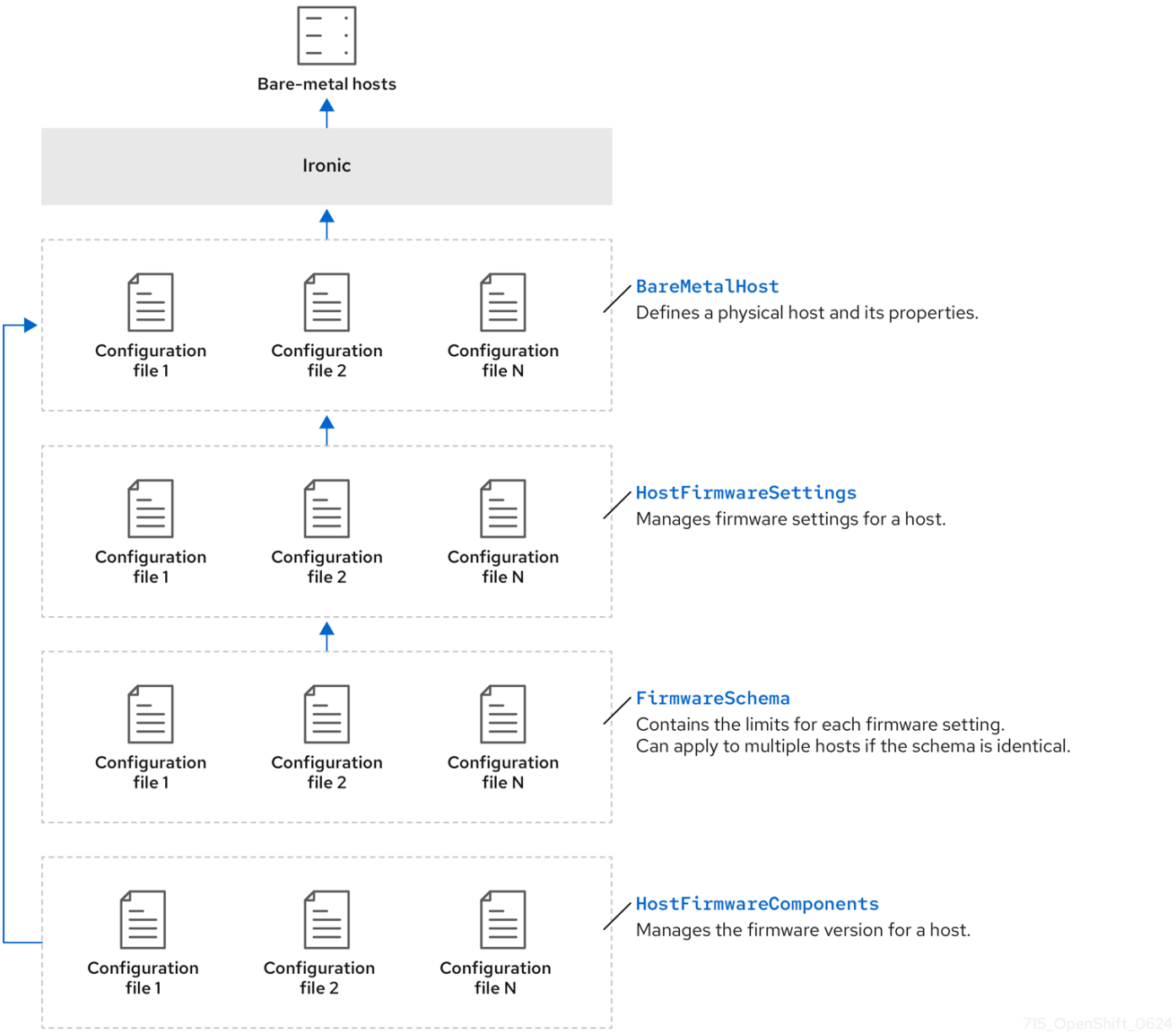
BMO は、Ironic API サービスを使用してクラスター内のベアメタルホストと接続します。Ironic サービスは、ホスト上のベースボード管理コントローラー (BMC) を使用して、マシンと接続します。

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
- ホストのファームウェアを特定のバージョンにアップグレードまたはダウングレードする

3.1.1. Bare Metal Operator のアーキテクチャー

Bare Metal Operator (BMO) は、次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。



BareMetalHost

BareMetalHost リソースは、物理ホストとそのプロパティーを定義します。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。

HostFirmwareComponents

Metal^3 は、BIOS およびベースボード管理コントローラー (BMC) ファームウェアのバージョンを記述する **HostFirmwareComponents** リソースを提供します。**HostFirmwareComponents** リソースの **spec** フィールドを編集することで、ホストのファームウェアを特定のバージョンにアップグレードまたはダウングレードできます。これは、特定のファームウェアバージョンに対してテストされた検証済みパターンを使用してデプロイする場合に便利です。

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 Metal^3 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

Metal^3 で、物理ホストとそのプロパティーを定義する **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。
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ーは監視されますが、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ーコアが複数の論理プロセッサー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** に設定されていて、外部管理に電源制御が不要な場合、フィールドは空のままでもかまいません。この設定では次のフィールドがサポートされています。 • **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 4.16 は、以下を含む BMC のハードウェア RAID をサポートします。 ● RAID レベル 0、1、5、6、および 10 をサポートする Fujitsu iRMC ● ファームウェアバージョン 6.10.30.20 以降および RAID レベル 0、1、および 5 に対応した Redfish API を使用する Dell iDRAC OpenShift Container Platform 4.16 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。

パラメーター	説明
	○ **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 4.16 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。

パラメーター	説明
	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。 spec: raid: hardwareRAIDVolume: [] 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は、**raid**、**hardwareRAIDVolumes** または **softwareRAIDVolumes** を nil 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る場合があります。

パラメーター	説明
rootDeviceHints: deviceName: hctl: model: vendor: serialNumber: minSizeGigabytes: wwn: wwnWithExtension: wwnVendorExtension: rotational:	**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。
hardware: cpu arch: model: clockMegahertz: flags: count:	**hardware.cpu** フィールドは、システム内のの 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。

パラメーター	説明
hardware: firmware:	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。
hardware: nics: - ip: name: mac: speedGbps: vlans: vlanId: pxe:	**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。
hardware: ramMebibytes:	ホストのメモリー容量 (MiB 単位)。
hardware: storage: - name: rotational: sizeBytes: serialNumber:	**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。
hardware: systemVendor: manufacturer: productName: serialNumber:	ホストの **manufacturer**、**productName**、および **serialNumber** に関する情報が含まれます。

パラメーター	説明
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。

パラメーター	説明
provisioning: state: id: image: raid: firmware: rootDeviceHints:	**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。 • **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**: Metal^3 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。

パラメーター	説明
	● **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。
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ーが含まれます。物理ホストのプロパティー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. ホストのリストを取得します。

```
$ 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
  online: true
  rootDeviceHints:
    deviceName: /dev/disk/by-id/scsi-<serial_number>
  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/disk/by-id/scsi-<serial_number>
      sizeBytes: 960197124096
      type: SSD
      vendor: ATA
    systemVendor:
      manufacturer: HPE
      productName: ProLiant DL380 Gen10 (868703-B21)
      serialNumber: CZ200606M3
  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/disk/by-id/scsi-<serial_number>
    state: provisioned
  triedCredentials:
    credentials:
      name: openshift-worker-0-bmc-secret
      namespace: openshift-machine-api
    credentialsVersion: "16120"
```

3.4. BAREMETALHOST リソースの編集

OpenShift Container Platform クラスターをベアメタルにデプロイした後、ノードの **BareMetalHost** リソースを編集する必要がある場合があります。たとえば、次のような例が考えられます。

- Assisted Installer を使用してクラスターをデプロイし、ベースボード管理コントローラー (BMC) のホスト名または IP アドレスを追加または編集する必要がある。
- ノードをプロビジョニング解除せずに、あるクラスターから別のクラスターに移動する必要がある。

前提条件

- ノードが **Provisioned**、**ExternallyProvisioned**、または **Available** 状態であることを確認する。

手順

1. ノードのリストを取得します。

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

2. ノードの **BareMetalHost** リソースを編集する前に、次のコマンドを実行してノードを Ironic からデタッチします。

```
$ oc annotate baremetalhost <node_name> -n openshift-machine-api
'baremetalhost.metal3.io/detached=true' (1)
```

(1) **<node_name>** はノード名に置き換えてください。

3. 次のコマンドを実行して、**BareMetalHost** リソースを編集します。

```
$ oc edit bmh <node_name> -n openshift-machine-api
```

4. 次のコマンドを実行して、ノードを Ironic に再アタッチします。

```
$ oc annotate baremetalhost <node_name> -n openshift-machine-api
'baremetalhost.metal3.io/detached'-
```

3.5. ブータブルでない ISO をベアメタルノードにアタッチする

DataImage リソースを使用すると、ブータブルでない汎用の ISO 仮想メディアイメージを、プロビジョニングされたノードにアタッチできます。リソースを適用すると、起動後にオペレーティングシステムから ISO イメージにアクセスできるようになります。これは、オペレーティングシステムをプロビジョニングした後、ノードが初めて起動する前にノードを設定する場合に便利です。

前提条件

- この機能をサポートするために、ノードが Redfish またはそれから派生したドライバーを使用している。
- ノードが **Provisioned** または **ExternallyProvisioned** 状態である。
- **name** が、**BareMetalHost** リソースで定義されているノードの名前と同じである。
- ISO イメージへの有効な **URL** がある。

手順

1. **DataImage** リソースを作成します。

```
apiVersion: metal3.io/v1alpha1
kind: DataImage
metadata:
  name: <node_name> (1)
spec:
  url: "http://dataimage.example.com/non-bootable.iso" (2)
```

(1) **BareMetalHost** リソースで定義されているノードの名前を指定します。

(2) ISO イメージへの URL とパスを指定します。

2. 次のコマンドを実行して、**DataImage** リソースをファイルに保存します。

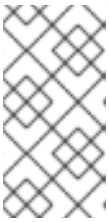
```
$ vim <node_name>-dataimage.yaml
```

3. 次のコマンドを実行して、**DataImage** リソースを適用します。

```
$ oc apply -f <node_name>-dataimage.yaml -n <node_namespace> (1)
```

(1) namespace が **BareMetalHost** リソースの namespace と一致するように **<node_namespace>** を置き換えます。たとえば、**openshift-machine-api** です。

4. ノードを再起動します。



注記

ノードを再起動するには、**reboot.metal3.io** アノテーションを割り当てるか、**BareMetalHost** リソースで **online** ステータスをリセットします。ベアメタルノードを強制的に再起動すると、ノードの状態がしばらくの間 **NotReady** に変わります。具体的には、5 分以上変わります。

5. 次のコマンドを実行して、**DataImage** リソースを表示します。

```
$ oc get dataimage <node_name> -n openshift-machine-api -o yaml
```

出力例

```
apiVersion: v1
items:
- apiVersion: metal3.io/v1alpha1
  kind: DataImage
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
        {"apiVersion":"metal3.io/v1alpha1","kind":"DataImage","metadata":{"annotations":
{},"name":"bmh-node-1","namespace":"openshift-machine-api"},"spec":
{"url":"http://dataimage.example.com/non-bootable.iso"}}
    creationTimestamp: "2024-06-10T12:00:00Z"
    finalizers:
    - dataimage.metal3.io
    generation: 1
    name: bmh-node-1
    namespace: openshift-machine-api
    ownerReferences:
    - apiVersion: metal3.io/v1alpha1
      blockOwnerDeletion: true
      controller: true
      kind: BareMetalHost
      name: bmh-node-1
      uid: 046cdf8e-0e97-485a-8866-e62d20e0f0b3
    resourceVersion: "21695581"
    uid: c5718f50-44b6-4a22-a6b7-71197e4b7b69
  spec:
    url: http://dataimage.example.com/non-bootable.iso
  status:
    attachedImage:
      url: http://dataimage.example.com/non-bootable.iso
    error:
      count: 0
      message: ""
    lastReconciled: "2024-06-10T12:05:00Z"
```

3.6. 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

3.6.1. 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 (1)
```

(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。

3.6.2. 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

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。

3.7. 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。

3.8. 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 (1)
```

(1) **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/delete-machine=true -n
openshift-machine-api
```

ここで、**<machine_name>** は削除するマシンの名前です。

7. ノードのリストを取得し、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。

3.9. 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 **FimwareSchema** 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。

3.10. 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

パラメーター	説明
<BIOS_setting_name> attribute_type: allowable_values: lower_bound: upper_bound: min_length: max_length: read_only: unique:	**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** の場合の下限値。 • **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。

3.11. 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。

3.12. HOSTFIRMWARECOMPONENTS リソースについて

Metal[3] は、BIOS およびベースボード管理コントローラー (BMC) ファームウェアのバージョンを記述する **HostFirmwareComponents** リソースを提供します。**HostFirmwareComponents** リソースには 2 つのセクションが含まれています。

1. **HostFirmwareComponents** spec
2. **HostFirmwareComponents** status

3.12.1. HostFirmwareComponents spec

HostFirmwareComponents リソースの **spec** セクションでは、ホストの BIOS および BMC バージョンの目的の状態を定義します。

表3.5 HostFirmwareComponents spec

パラメーター	説明
`updates:` `  component:` `  url:`	**updates** 設定には、更新するコンポーネントを含めます。フィールドの詳細は以下のとおりです。 ● **component**: コンポーネントの名前。有効な設定は **bios** または **bmc** です。 ● **url**: コンポーネントのファームウェア仕様とバージョンへの URL。

3.12.2. HostFirmwareComponents status

HostFirmwareComponents リソースの **status** セクションは、ホストの BIOS および BMC バージョンの現在のステータスを返します。

表3.6 HostFirmwareComponents status

パラメーター	説明

パラメーター	説明
components: component: initialVersion: currentVersion: lastVersionFlashed: updatedAt:	**components** セクションには、コンポーネントのステータスが含まれています。フィールドの詳細は以下のとおりです。 • **component**: ファームウェアコンポーネントの名前。**bios** または **bmc** を返します。 • **initialVersion**: コンポーネントの初期ファームウェアバージョン。Ironic は、**BareMetalHost** リソースを作成するときにこの情報を取得します。ユーザーが変更することはできません。 • **currentVersion**: コンポーネントの現在のファームウェアバージョン。この値は、Ironic がベアメタルホストのファームウェアを更新するまで、**initialVersion** の値と同じです。 • **lastVersionFlashed**: ベアメタルホストでフラッシュされたコンポーネントの最後のファームウェアバージョン。Ironic がファームウェアを更新するまで、このフィールドは **null** を返します。 • **updatedAt**: Ironic がベアメタルホストのファームウェアを更新したときのタイムスタンプ。
updates: component: url:	**updates** 設定には、更新されたコンポーネントが含まれています。フィールドの詳細は以下のとおりです。 • **component**: コンポーネントの名前。 • **url**: コンポーネントのファームウェア仕様とバージョンへの URL。

3.13. HOSTFIRMWARECOMPONENTS リソースの取得

HostFirmwareComponents リソースには、物理ホストの BIOS およびベースボード管理コントローラー (BMC) の特定のファームウェアバージョンが含まれています。ファームウェアのバージョンとステータスを確認するには、物理ホストの **HostFirmwareComponents** リソースを取得する必要があります。

手順

1. **HostFirmwareComponents** リソースの詳細なリストを取得します。

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

2. **HostFirmwareComponents** リソースのリストを取得します。

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

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

```
$ oc get hostfirmwarecomponents <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: HostFirmwareComponents
metadata:
  creationTimestamp: 2024-04-25T20:32:06Z"
  generation: 1
  name: ostest-master-2
  namespace: openshift-machine-api
  ownerReferences:
  - apiVersion: metal3.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: BareMetalHost
    name: ostest-master-2
    uid: 16022566-7850-4dc8-9e7d-f216211d4195
  resourceVersion: "2437"
  uid: 2038d63f-afc0-4413-8ffe-2f8e098d1f6c
spec:
  updates: []
status:
  components:
  - component: bios
    currentVersion: 1.0.0
    initialVersion: 1.0.0
  - component: bmc
    currentVersion: "1.00"
    initialVersion: "1.00"
  conditions:
  - lastTransitionTime: "2024-04-25T20:32:06Z"
    message: ""
    observedGeneration: 1
    reason: OK
    status: "True"
    type: Valid
  - lastTransitionTime: "2024-04-25T20:32:06Z"
    message: ""
    observedGeneration: 1
    reason: OK
    status: "False"
    type: ChangeDetected
  lastUpdated: "2024-04-25T20:32:06Z"
  updates: []
```

3.14. HOSTFIRMWARECOMPONENTS リソースの編集

ノードの **HostFirmwareComponents** リソースを編集できます。

手順

1. **HostFirmwareComponents** リソースの詳細なリストを取得します。

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

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

```
$ oc edit <host_name> hostfirmwarecomponents -n openshift-machine-api (1)
```

(1) ここで、**<host_name>** はホストの名前です。**HostFirmwareComponents** リソースが、ターミナルのデフォルトのエディターで開きます。

出力例

```
---
apiVersion: metal3.io/v1alpha1
kind: HostFirmwareComponents
metadata:
  creationTimestamp: 2024-04-25T20:32:06Z"
  generation: 1
  name: ostest-master-2
  namespace: openshift-machine-api
  ownerReferences:
  - apiVersion: metal3.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: BareMetalHost
    name: ostest-master-2
    uid: 16022566-7850-4dc8-9e7d-f216211d4195
  resourceVersion: "2437"
  uid: 2038d63f-afc0-4413-8ffe-2f8e098d1f6c
spec:
  updates:
    - name: bios (1)
      url: https://myurl.with.firmware.for.bios (2)
    - name: bmc (3)
      url: https://myurl.with.firmware.for.bmc (4)
status:
  components:
  - component: bios
    currentVersion: 1.0.0
    initialVersion: 1.0.0
  - component: bmc
    currentVersion: "1.00"
    initialVersion: "1.00"
  conditions:
  - lastTransitionTime: "2024-04-25T20:32:06Z"
    message: ""
    observedGeneration: 1
    reason: OK
```

```
    status: "True"
    type: Valid
  - lastTransitionTime: "2024-04-25T20:32:06Z"
    message: ""
    observedGeneration: 1
    reason: OK
    status: "False"
    type: ChangeDetected
  lastUpdated: "2024-04-25T20:32:06Z"
```

1. BIOS のバージョンを設定するには、**name** 属性を **bios** に設定します。
2. BIOS のバージョンを設定するには、**url** 属性を BIOS のファームウェアバージョンの URL に設定します。
3. BMC のバージョンを設定するには、**name** 属性を **bmc** に設定します。
4. BMC のバージョンを設定するには、**url** 属性を BMC のファームウェアバージョンの URL に設定します。

3. 変更を保存し、エディターを終了します。
4. ホストのマシン名を取得します。

```
$ oc get bmh <host_name> -n openshift-machine name (1)
```

1. 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。

5. 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/delete-machine=true -n
openshift-machine-api (1)
```

1. ここで、**<machine_name>** は削除するマシンの名前です。

6. ノードのリストを取得し、ワーカーノードの数をカウントします。

```
$ oc get nodes
```

7. マシンセットを取得します。

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

8. 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> (1)
```

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

9. ホストが **Available** 状態になったら、マシンセットをスケールアップして、**HostFirmwareComponents** リソースの変更を有効にします。

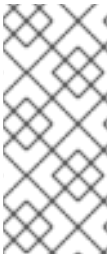
```
$ oc scale machineset <machineset_name> -n openshift-machine-api --replicas=<n>
```
(1)

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

第4章 OPENSHIFT クラスターでのマルチアーキテクチャーのコンピュートマシンの設定

4.1. マルチアーキテクチャーのコンピュートマシンを含むクラスターについて

マルチアーキテクチャー計算マシンを使用する OpenShift Container Platform クラスターは、異なるアーキテクチャーのコンピュートマシンをサポートするクラスターです。



注記

クラスター内に複数のアーキテクチャーを持つノードがある場合、イメージのアーキテクチャーはノードのアーキテクチャーと一致している必要があります。Pod が適切なアーキテクチャーを持つノードに割り当てられていること、およびそれがイメージアーキテクチャーと一致していることを確認する必要があります。ノードへの Pod の割り当ての詳細は、ノードへの Pod の割り当て を参照してください。

重要

Cluster Samples Operator は、マルチアーキテクチャーのコンピューティングマシンを備えたクラスターではサポートされません。この機能がなくてもクラスターを作成できます。詳細は、クラスター機能の有効化 を参照してください。

単一アーキテクチャーのクラスターをマルチアーキテクチャーのコンピューティングマシンをサポートするクラスターに移行する方法は、マルチアーキテクチャーのコンピューティングマシン を含むクラスターへ の移行 を参照してください。

4.1.1. マルチアーキテクチャーのコンピュートマシンを使用したクラスターの設定

各種のインストールオプションとプラットフォームを使用してマルチアーキテクチャーコンピュートマシンを含むクラスターを作成するには、次の表のドキュメントを使用してください。

表4.1 マルチアーキテクチャーのコンピュートマシンを含むクラスターのインストールオプション

ドキュメントのセクション	プラットフォーム	user-provisioned installation	installer-provisioned installation	コントロールプレーン	コンピュートノード
Azure 上でマルチアーキテクチャーのコンピューティングマシンを含むクラスターを作成する	Microsoft Azure		✓	**aarch64** または **x86_64**	**aarch64** 、**x86_64**
AWS 上でマルチアーキテクチャーのコンピューティングマシンを含むクラスターを作成する	Amazon Web Services (AWS)		✓	**aarch64** または **x86_64**	**aarch64** 、**x86_64**

ドキュメントのセクション	プラットフォーム	user-provisioned installation	installer-provisioned installation	コントロールプレーン	コンピュートノード
GCP 上でマルチアーキテクチャーのコンピューティングマシンを含むクラスターを作成する	Google Cloud Platform (GCP)		✓	**aarch64** または **x86_64**	**aarch64** 、**x86_64**
ベアメタル、IBM Power、または IBM Z 上でマルチアーキテクチャーのコンピューティングマシンを含むクラスターを作成する	ベアメタル	✓		**aarch64** または **x86_64**	**aarch64** 、**x86_64**
	IBM Power	✓		**x86_64** または **ppc64le**	**x86_64**、**ppc64le**
	IBM Z	✓		**x86_64** または **s390x**	**x86_64**、**s390x**
z/VM を使用した IBM Z® および IBM® LinuxONE 上でマルチアーキテクチャーのコンピュートマシンを含むクラスターを作成する	IBM Z® および IBM® LinuxONE	✓		**x86_64**	**x86_64**、**s390x**
RHEL KVM を使用した IBM Z® および IBM® LinuxONE 上でマルチアーキテクチャーのコンピュートマシンを含むクラスターを作成する	IBM Z® および IBM® LinuxONE	✓		**x86_64**	**x86_64**、**s390x**
IBM Power® 上でマルチアーキテクチャーのコンピュートマシンを含むクラスターを作成する	IBM Power®	✓		**x86_64**	**x86_64**、**ppc64le**

重要

ゼロからの自動スケーリングは現在、Google Cloud Platform (GCP) ではサポートされていません。

4.2. AZURE でマルチアーキテクチャーコンピューティングマシンを使用したクラスターを作成する

マルチアーキテクチャーコンピューティングマシンを使用して Azure クラスターをデプロイするには、まず、マルチアーキテクチャーインストーラーバイナリーを使用する単一アーキテクチャーの Azure インストーラープロビジョニングクラスターを作成する必要があります。Azure インストールの詳細については、カスタマイズを使用した Azure へのクラスターのインストール を参照してください。

シングルアーキテクチャーのコンピュートマシンを持つ現在のクラスターを、マルチアーキテクチャーのコンピュートマシンを持つクラスターに移行することもできます。詳細は、マルチアーキテクチャーのコンピューティングマシンを使用したクラスターへの移行 を 参照してください。

マルチアーキテクチャークラスターを作成した後、異なるアーキテクチャーのノードをクラスターに追加できます。

4.2.1. クラスターの互換性の確認

異なるアーキテクチャーのコンピュートノードをクラスターに追加する前に、クラスターがマルチアーキテクチャー互換であることを確認する必要があります。

前提条件

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

手順

1. OpenShift CLI (**oc**) にログインします。
2. 次のコマンドを実行すると、クラスターがアーキテクチャーペイロードを使用していることを確認できます。

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

検証

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用しています。

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

その後、クラスターへのマルチアーキテクチャーコンピュートノードの追加を開始できます。

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用していません。

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

重要

クラスターがマルチアーキテクチャーコンピューティングマシンをサポートするようにクラスターを移行するには、マルチアーキテクチャーコンピューティングマシンを 使用したクラスターへ の移行 の手順に従います。

4.2.2. Azure イメージギャラリーを使用して 64 ビット ARM ブートイメージを作成する

次の手順では、64 ビット ARM ブートイメージを手動で生成する方法について説明します。

前提条件

- Azure CLI (**az**) をインストールしている。
- マルチアーキテクチャーインストーラーバイナリーを使用して、単一アーキテクチャーの Azure インストーラープロビジョニングクラスターを作成している。

手順

1. Azure アカウントにログインします。

```
$ az login
```

2. ストレージアカウントを作成し、**aarch64** 仮想ハードディスク (VHD) をストレージアカウントにアップロードします。OpenShift Container Platform インストールプログラムはリソースグループを作成しますが、ブートイメージをカスタムの名前付きリソースグループにアップロードすることもできます。

```
$ az storage account create -n ${STORAGE_ACCOUNT_NAME} -g
${RESOURCE_GROUP} -l westus --sku Standard_LRS 1
```

1 **westus** オブジェクトはリージョンの例です。

3. 生成したストレージアカウントを使用してストレージコンテナーを作成します。

```
$ az storage container create -n ${CONTAINER_NAME} --account-name
${STORAGE_ACCOUNT_NAME}
```

4. URL と **ARM64** VHD 名を抽出するには、OpenShift Container Platform インストールプログラムの JSON ファイルを使用する必要があります。

   a. 次のコマンドを実行して、**URL** フィールドを抽出し、ファイル名として **RHCOS_VHD_ORIGIN_URL** に設定します。

```
$ RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get
configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r
'.architectures.aarch64."rhel-coreos-extensions"."azure-disk".url')
```

   b. 次のコマンドを実行して、**aarch64** VHD 名を抽出し、ファイル名として **BLOB_NAME** に設定します。

```
$ BLOB_NAME=rhcos-$(oc -n openshift-machine-config-operator get configmap/coreos-
bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.aarch64."rhel-coreos-
extensions"."azure-disk".release')-azure.aarch64.vhd
```

5. Shared Access Signature (SAS) トークンを生成します。このトークンを使用して、次のコマンドで RHCOS VHD をストレージコンテナーにアップロードします。

```
$ end=`date -u -d "30 minutes" '+%Y-%m-%dT%H:%MZ'`
```

```
$ sas=`az storage container generate-sas -n ${CONTAINER_NAME} --account-name
${STORAGE_ACCOUNT_NAME} --https-only --permissions dlrw --expiry $end -o tsv`
```

6. RHCOS VHD をストレージコンテナーにコピーします。

```
$ az storage blob copy start --account-name ${STORAGE_ACCOUNT_NAME} --sas-token
"$sas" \
 --source-uri "${RHCOS_VHD_ORIGIN_URL}" \
 --destination-blob "${BLOB_NAME}" --destination-container ${CONTAINER_NAME}
```

次のコマンドを使用して、コピープロセスのステータスを確認できます。

```
$ az storage blob show -c ${CONTAINER_NAME} -n ${BLOB_NAME} --account-name
${STORAGE_ACCOUNT_NAME} | jq .properties.copy
```

出力例

```
{
 "completionTime": null,
 "destinationSnapshot": null,
 "id": "1fd97630-03ca-489a-8c4e-cfe839c9627d",
 "incrementalCopy": null,
 "progress": "17179869696/17179869696",
 "source": "https://rhcos.blob.core.windows.net/imagebucket/rhcos-411.86.202207130959-0-
azure.aarch64.vhd",
 "status": "success", (1)
 "statusDescription": null
}
```

(1) status パラメーターに **success** オブジェクトが表示されたら、コピープロセスは完了です。

7. 次のコマンドを使用してイメージギャラリーを作成します。

```
$ az sig create --resource-group ${RESOURCE_GROUP} --gallery-name
${GALLERY_NAME}
```

イメージギャラリーを使用してイメージ定義を作成します。次のコマンド例では、**rhcos-arm64** がイメージ定義の名前です。

```
$ az sig image-definition create --resource-group ${RESOURCE_GROUP} --gallery-name
${GALLERY_NAME} --gallery-image-definition rhcos-arm64 --publisher RedHat --offer arm -
-sku arm64 --os-type linux --architecture Arm64 --hyper-v-generation V2
```

8. VHD の URL を取得してファイル名として **RHCOS_VHD_URL** に設定するには、次のコマンドを実行します。

```
$ RHCOS_VHD_URL=$(az storage blob url --account-name
${STORAGE_ACCOUNT_NAME} -c ${CONTAINER_NAME} -n "${BLOB_NAME}" -o tsv)
```

9. **RHCOS_VHD_URL** ファイル、ストレージアカウント、リソースグループ、およびイメージギャラリーを使用して、イメージバージョンを作成します。次の例では、**1.0.0** がイメージバージョンです。

```
$ az sig image-version create --resource-group ${RESOURCE_GROUP} --gallery-name
```

```
${GALLERY_NAME} --gallery-image-definition rhcos-arm64 --gallery-image-version 1.0.0 --
os-vhd-storage-account ${STORAGE_ACCOUNT_NAME} --os-vhd-uri
${RHCOS_VHD_URL}
```

10. **arm64** ブートイメージが生成されました。次のコマンドを使用して、イメージの ID にアクセスできます。

```
$ az sig image-version show -r $GALLERY_NAME -g $RESOURCE_GROUP -i rhcos-arm64
-e 1.0.0
```

次の例のイメージ ID は、コンピュートマシンセットの **recourseID** パラメーターで使用されます。

resourceID の例

```
/resourceGroups/${RESOURCE_GROUP}/providers/Microsoft.Compute/galleries/${GALLERY
_NAME}/images/rhcos-arm64/versions/1.0.0
```

4.2.3. Azure イメージギャラリーを使用して 64 ビット x86 ブートイメージを作成する

次の手順では、64 ビット x86 ブートイメージを手動で生成する方法について説明します。

前提条件

- Azure CLI (**az**) をインストールしている。
- マルチアーキテクチャーインストーラーバイナリーを使用して、単一アーキテクチャーの Azure インストーラープロビジョニングクラスターを作成している。

手順

1. 次のコマンドを実行して、Azure アカウントにログインします。

```
$ az login
```

2. ストレージアカウントを作成し、次のコマンドを実行して **x86_64** 仮想ハードディスク (VHD) をストレージアカウントにアップロードします。OpenShift Container Platform インストールプログラムがリソースグループを作成します。なお、ブートイメージは、カスタム名のリソースグループにアップロードすることもできます。

```
$ az storage account create -n ${STORAGE_ACCOUNT_NAME} -g
${RESOURCE_GROUP} -l westus --sku Standard_LRS (1)
```

(1) **westus** オブジェクトはリージョンの例です。

3. 次のコマンドを実行して、生成したストレージアカウントを使用してストレージコンテナーを作成します。

```
$ az storage container create -n ${CONTAINER_NAME} --account-name
${STORAGE_ACCOUNT_NAME}
```

4. OpenShift Container Platform インストールプログラムの JSON ファイルを使用して、URL と **x86_64** VHD 名を抽出します。

   a. 次のコマンドを実行して、**URL** フィールドを抽出し、ファイル名として **RHCOS_VHD_ORIGIN_URL** に設定します。

   ```
   $ RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.x86_64."rhel-coreos-extensions"."azure-disk".url')
   ```

   b. 次のコマンドを実行して、**x86_64** VHD 名を抽出し、ファイル名として **BLOB_NAME** に設定します。

   ```
   $ BLOB_NAME=rhcos-$(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.x86_64."rhel-coreos-extensions"."azure-disk".release')-azure.x86_64.vhd
   ```

5. Shared Access Signature (SAS) トークンを生成します。このトークンを使用して、次のコマンドを実行し、RHCOS VHD をストレージコンテナーにアップロードします。

   ```
   $ end=`date -u -d "30 minutes" '+%Y-%m-%dT%H:%MZ'`
   ```

   ```
   $ sas=`az storage container generate-sas -n ${CONTAINER_NAME} --account-name ${STORAGE_ACCOUNT_NAME} --https-only --permissions dlrw --expiry $end -o tsv`
   ```

6. 次のコマンドを実行して、RHCOS VHD をストレージコンテナーにコピーします。

   ```
   $ az storage blob copy start --account-name ${STORAGE_ACCOUNT_NAME} --sas-token "$sas" \
    --source-uri "${RHCOS_VHD_ORIGIN_URL}" \
    --destination-blob "${BLOB_NAME}" --destination-container ${CONTAINER_NAME}
   ```

   次のコマンドを実行すると、コピープロセスのステータスを確認できます。

   ```
   $ az storage blob show -c ${CONTAINER_NAME} -n ${BLOB_NAME} --account-name ${STORAGE_ACCOUNT_NAME} | jq .properties.copy
   ```

   出力例

   ```
   {
    "completionTime": null,
    "destinationSnapshot": null,
    "id": "1fd97630-03ca-489a-8c4e-cfe839c9627d",
    "incrementalCopy": null,
    "progress": "17179869696/17179869696",
    "source": "https://rhcos.blob.core.windows.net/imagebucket/rhcos-411.86.202207130959-0-azure.aarch64.vhd",
    "status": "success", (1)
    "statusDescription": null
   }
   ```

   (1) **status** パラメーターに **success** オブジェクトが表示されたら、コピープロセスは完了です。

7. 次のコマンドを実行してイメージギャラリーを作成します。

```
$ az sig create --resource-group ${RESOURCE_GROUP} --gallery-name
${GALLERY_NAME}
```

8. 次のコマンドを実行して、イメージギャラリーを使用してイメージ定義を作成します。

```
$ az sig image-definition create --resource-group ${RESOURCE_GROUP} --gallery-name
${GALLERY_NAME} --gallery-image-definition rhcos-x86_64 --publisher RedHat --offer
x86_64 --sku x86_64 --os-type linux --architecture x64 --hyper-v-generation V2
```

このコマンド例の **rhcos-x86_64** は、イメージ定義の名前です。

9. VHD の URL を取得してファイル名として **RHCOS_VHD_URL** に設定するには、次のコマンドを実行します。

```
$ RHCOS_VHD_URL=$(az storage blob url --account-name
${STORAGE_ACCOUNT_NAME} -c ${CONTAINER_NAME} -n "${BLOB_NAME}" -o tsv)
```

10. 次のコマンドを実行して、**RHCOS_VHD_URL** ファイル、ストレージアカウント、リソースグループ、イメージギャラリーを使用してイメージバージョンを作成します。

```
$ az sig image-version create --resource-group ${RESOURCE_GROUP} --gallery-name
${GALLERY_NAME} --gallery-image-definition rhcos-arm64 --gallery-image-version 1.0.0 --
os-vhd-storage-account ${STORAGE_ACCOUNT_NAME} --os-vhd-uri
${RHCOS_VHD_URL}
```

この例では、**1.0.0** がイメージバージョンです。

11. オプション: 次のコマンドを実行して、生成された **x86_64** ブートイメージの ID にアクセスします。

```
$ az sig image-version show -r $GALLERY_NAME -g $RESOURCE_GROUP -i rhcos-
x86_64 -e 1.0.0
```

次の例のイメージ ID は、コンピュートマシンセットの **recourseID** パラメーターで使用されます。

resourceID の例

```
/resourceGroups/${RESOURCE_GROUP}/providers/Microsoft.Compute/galleries/${GALLERY
_NAME}/images/rhcos-x86_64/versions/1.0.0
```

4.2.4. Azure クラスターにマルチアーキテクチャーコンピュートマシンセットを追加する

マルチアーキテクチャークラスターを作成した後、異なるアーキテクチャーのノードを追加できます。

マルチアーキテクチャーコンピュートマシンをマルチアーキテクチャークラスターに追加する方法には、次のものがあります。

- 64 ビット ARM コントロールプレーンマシンを使用し、すでに 64 ビット ARM コンピュートマシンが含まれているクラスターに 64 ビット x86 コンピュートマシンを追加します。この場合、64 ビット x86 がセカンダリーアーキテクチャーと見なされます。
- 64 ビット x86 コントロールプレーンマシンを使用し、すでに 64 ビット x86 コンピュートマシンが含まれているクラスターに 64 ビット ARM コンピュートマシンを追加します。この場合、64 ビット ARM がセカンダリーアーキテクチャーと見なされます。

Azure でカスタムコンピュートマシンセットを作成するには、「Azure でのコンピュートマシンセットの作成」を参照してください。

注記

セカンダリーアーキテクチャーノードをクラスターに追加する前に、Multiarch Tuning Operator をインストールし、**ClusterPodPlacementConfig** カスタムリソースをデプロイすることを推奨します。詳細は、「Multiarch Tuning Operator を使用してマルチアーキテクチャークラスター上のワークロードを管理する」を参照してください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- 64 ビット ARM または 64 ビット x86 ブートイメージを作成した。
- インストールプログラムを使用して、マルチアーキテクチャーインストーラーバイナリーを使用する 64 ビット ARM または 64 ビット x86 シングルアーキテクチャー Azure クラスターを作成した。

手順

1. OpenShift CLI (**oc**) にログインします。
2. YAML ファイルを作成し、設定を追加して、クラスター内の 64 ビット ARM または 64 ビット x86 コンピュートノードを制御するコンピュートマシンセットを作成します。

Azure の 64 ビット ARM または 64 ビット x86 コンピュートノードの MachineSet オブジェクトの例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id>
    machine.openshift.io/cluster-api-machine-role: worker
    machine.openshift.io/cluster-api-machine-type: worker
  name: <infrastructure_id>-machine-set-0
  namespace: openshift-machine-api
spec:
  replicas: 2
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-machine-set-0
  template:
    metadata:
```

```
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-machine-set-0
    spec:
      lifecycleHooks: {}
      metadata: {}
      providerSpec:
        value:
          acceleratedNetworking: true
          apiVersion: machine.openshift.io/v1beta1
          credentialsSecret:
            name: azure-cloud-credentials
            namespace: openshift-machine-api
          image:
            offer: ""
            publisher: ""
            resourceID:
/resourceGroups/${RESOURCE_GROUP}/providers/Microsoft.Compute/galleries/${GALLERY
_NAME}/images/rhcos-arm64/versions/1.0.0 (1)
            sku: ""
            version: ""
          kind: AzureMachineProviderSpec
          location: <region>
          managedIdentity: <infrastructure_id>-identity
          networkResourceGroup: <infrastructure_id>-rg
          osDisk:
            diskSettings: {}
            diskSizeGB: 128
            managedDisk:
              storageAccountType: Premium_LRS
            osType: Linux
          publicIP: false
          publicLoadBalancer: <infrastructure_id>
          resourceGroup: <infrastructure_id>-rg
          subnet: <infrastructure_id>-worker-subnet
          userDataSecret:
            name: worker-user-data
          vmSize: Standard_D4ps_v5 (2)
          vnet: <infrastructure_id>-vnet
          zone: "<zone>"
```

(1) **resourceID** パラメーターを **arm64** または **amd64** ブートイメージに設定します。

(2) **vmSize** パラメーターを、インストールで使用されているインスタンスタイプに設定します。インスタンスタイプの例として、**Standard_D4ps_v5** または **D8ps** があります。

3. 次のコマンドを実行してコンピュートマシンセットを作成します。

```
$ oc create -f <file_name> (1)
```

(1) **<file_name>** は、コンピュートマシン設定を含む YAML ファイルの名前に置き換えます。たとえば、**arm64-machine-set-0.yaml**、または **amd64-machine-set-0.yaml** です。

検証

1. 次のコマンドを実行して、新しいマシンが実行中であることを確認します。

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

出力に、作成したマシンセットが含まれている必要があります。

出力例

```
NAME                                 DESIRED  CURRENT  READY  AVAILABLE  AGE
<infrastructure_id>-machine-set-0    2        2        2      2          10m
```

2. 次のコマンドを実行すると、ノードが準備完了状態でスケジュール可能かどうかを確認できます。

```
$ oc get nodes
```

関連情報

- Azure でコンピュートマシンセットを作成
- Multiarch Tuning Operator を使用してマルチアーキテクチャークラスター上のワークロードを管理する

4.3. AWS 上でマルチアーキテクチャーのコンピューティングマシンを含むクラスターを作成する

マルチアーキテクチャーのコンピューティングマシンを含む AWS クラスターを作成するには、まずマルチアーキテクチャーインストーラーバイナリーを使用して、単一アーキテクチャーの AWS インストーラーによってプロビジョニングされたクラスターを作成する必要があります。AWS のインストールの詳細は、カスタマイズを使用した AWS へのクラスターのインストール を参照してください。

シングルアーキテクチャーのコンピュートマシンを持つ現在のクラスターを、マルチアーキテクチャーのコンピュートマシンを持つクラスターに移行することもできます。詳細は、マルチアーキテクチャーのコンピューティングマシンを使用したクラスターへの移行 を 参照してください。

マルチアーキテクチャークラスターを作成した後、異なるアーキテクチャーのノードをクラスターに追加できます。

4.3.1. クラスターの互換性の確認

異なるアーキテクチャーのコンピュートノードをクラスターに追加する前に、クラスターがマルチアーキテクチャー互換であることを確認する必要があります。

前提条件

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

手順

1. OpenShift CLI (**oc**) にログインします。

2. 次のコマンドを実行すると、クラスターがアーキテクチャーペイロードを使用していることを確認できます。

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

検証

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用しています。

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

その後、クラスターへのマルチアーキテクチャーコンピュートノードの追加を開始できます。

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用していません。

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

重要

クラスターがマルチアーキテクチャーコンピューティングマシンをサポートするようにクラスターを移行するには、マルチアーキテクチャーコンピューティングマシンを 使用したクラスターへ の移行 の手順に従います。

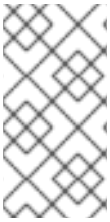
4.3.2. AWS クラスターにマルチアーキテクチャーコンピュートマシンセットを追加する

マルチアーキテクチャークラスターを作成した後、異なるアーキテクチャーのノードを追加できます。

マルチアーキテクチャーコンピュートマシンをマルチアーキテクチャークラスターに追加する方法には、次のものがあります。

- 64 ビット ARM コントロールプレーンマシンを使用し、すでに 64 ビット ARM コンピュートマシンが含まれているクラスターに 64 ビット x86 コンピュートマシンを追加します。この場合、64 ビット x86 がセカンダリーアーキテクチャーと見なされます。

- 64 ビット x86 コントロールプレーンマシンを使用し、すでに 64 ビット x86 コンピュートマシンが含まれているクラスターに 64 ビット ARM コンピュートマシンを追加します。この場合、64 ビット ARM がセカンダリーアーキテクチャーと見なされます。



注記

セカンダリーアーキテクチャーノードをクラスターに追加する前に、Multiarch Tuning Operator をインストールし、**ClusterPodPlacementConfig** カスタムリソースをデプロイすることを推奨します。詳細は、「Multiarch Tuning Operator を使用してマルチアーキテクチャークラスター上のワークロードを管理する」を参照してください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- インストールプログラムを使用して、マルチアーキテクチャーインストーラーバイナリーを使用する 64 ビット ARM または 64 ビット x86 シングルアーキテクチャー AWS クラスターを作成した。

手順

1. OpenShift CLI (**oc**) にログインします。
2. YAML ファイルを作成し、設定を追加して、クラスター内の 64 ビット ARM または 64 ビット x86 コンピュートノードを制御するコンピュートマシンセットを作成します。

AWS の 64 ビット ARM または x86 コンピュートノードの MachineSet オブジェクトの例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-aws-machine-set-0 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id> 3
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>-<zone> 4
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: <role> 5
        machine.openshift.io/cluster-api-machine-type: <role> 6
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>-<zone> 7
    spec:
      metadata:
        labels:
          node-role.kubernetes.io/<role>: ""
      providerSpec:
        value:
          ami:
            id: ami-02a574449d4f4d280 8
          apiVersion: awsproviderconfig.openshift.io/v1beta1
          blockDevices:
            - ebs:
                iops: 0
                volumeSize: 120
                volumeType: gp2
          credentialsSecret:
            name: aws-cloud-credentials
          deviceIndex: 0
```

```
        iamInstanceProfile:
          id: <infrastructure_id>-worker-profile 9
        instanceType: m6g.xlarge 10
        kind: AWSMachineProviderConfig
        placement:
          availabilityZone: us-east-1a 11
          region: <region> 12
        securityGroups:
          - filters:
              - name: tag:Name
                values:
                  - <infrastructure_id>-worker-sg 13
        subnet:
          filters:
            - name: tag:Name
              values:
                - <infrastructure_id>-private-<zone>
        tags:
          - name: kubernetes.io/cluster/<infrastructure_id> 14
            value: owned
          - name: <custom_tag_name>
            value: <custom_tag_value>
        userDataSecret:
          name: worker-user-data
```

1 2 3 9 13 14 クラスターのプロビジョニング時に設定したクラスター ID を基にするインフラストラクチャー ID を指定します。OpenShift CLI (**oc**) がインストールされている場合は、以下のコマンドを実行してインフラストラクチャー ID を取得できます。

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructure cluster
```

4 7 インフラストラクチャー ID、ロールノードラベル、およびゾーンを指定します。

5 6 追加するロールノードラベルを指定します。

8 ノードの AWS ゾーンに Red Hat Enterprise Linux CoreOS (RHCOS) Amazon Machine Image (AMI) を指定します。RHCOS AMI はマシンのアーキテクチャーと互換性がある必要があります。

```
$ oc get configmap/coreos-bootimages \
  -n openshift-machine-config-operator \
  -o jsonpath='{.data.stream}' | jq \
  -r '.architectures.<arch>.images.aws.regions."<region>".image'
```

10 選択した AMI の CPU アーキテクチャーに合ったマシンタイプを指定します。詳細は、「AWS 64 ビット ARM のテスト済みインスタンスタイプ」を参照してください。

11 ゾーンを指定します。たとえば、**us-east-1a** です。選択したゾーンに必要なアーキテクチャーを備えたマシンがあることを確認してください。

12 リージョンを指定します。たとえば、**us-east-1** などです。選択したゾーンに必要なアーキテクチャーを備えたマシンがあることを確認してください。

3. 次のコマンドを実行してコンピュートマシンセットを作成します。

```
$ oc create -f <file_name> 1
```

1 **<file_name>** は、コンピュートマシン設定を含む YAML ファイルの名前に置き換えます。たとえば、**aws-arm64-machine-set-0.yaml**、または **aws-amd64-machine-set-0.yaml** です。

検証

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

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

出力に、作成したマシンセットが含まれている必要があります。

出力例

```
NAME                                DESIRED  CURRENT  READY  AVAILABLE  AGE
<infrastructure_id>-aws-machine-set-0    2        2      2          2  10m
```

2. 次のコマンドを実行すると、ノードが準備完了状態でスケジュール可能かどうかを確認できます。

```
$ oc get nodes
```

関連情報

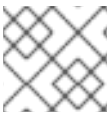
- AWS 64 ビット ARM のテスト済みインスタンスタイプ
- Multiarch Tuning Operator を使用してマルチアーキテクチャークラスター上のワークロードを管理する

4.4. GCP 上でマルチアーキテクチャーのコンピューティングマシンを含むクラスターを作成する

マルチアーキテクチャーのコンピュートマシンを含む Google Cloud Platform (GCP) クラスターを作成するには、まず、マルチアーキテクチャーインストーラーバイナリーを使用して、単一アーキテクチャーの GCP インストーラーによってプロビジョニングされたクラスターを作成する必要があります。AWS のインストールの詳細は、カスタマイズを使用した GCP へのクラスターのインストール を参照してください。

シングルアーキテクチャーのコンピュートマシンを持つ現在のクラスターを、マルチアーキテクチャーのコンピュートマシンを持つクラスターに移行することもできます。詳細は、マルチアーキテクチャーのコンピューティングマシンを使用したクラスターへの移行 を 参照してください。

マルチアーキテクチャークラスターを作成した後、異なるアーキテクチャーのノードをクラスターに追加できます。



注記

GCP の 64 ビット ARM マシンでは、セキュアブートは現在サポートされていません。

4.4.1. クラスターの互換性の確認

異なるアーキテクチャーのコンピュートノードをクラスターに追加する前に、クラスターがマルチアーキテクチャー互換であることを確認する必要があります。

前提条件

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

手順

1. OpenShift CLI (**oc**) にログインします。
2. 次のコマンドを実行すると、クラスターがアーキテクチャーペイロードを使用していることを確認できます。

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

検証

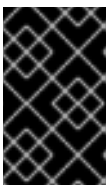
- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用しています。

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

その後、クラスターへのマルチアーキテクチャーコンピュートノードの追加を開始できます。

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用していません。

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```



重要

クラスターがマルチアーキテクチャーコンピューティングマシンをサポートするようにクラスターを移行するには、マルチアーキテクチャーコンピューティングマシンを 使用したクラスターへ の移行 の手順に従います。

4.4.2. GCP クラスターにマルチアーキテクチャーコンピュートマシンセットを追加する

マルチアーキテクチャークラスターを作成した後、異なるアーキテクチャーのノードを追加できます。

マルチアーキテクチャーコンピュートマシンをマルチアーキテクチャークラスターに追加する方法には、次のものがあります。

- 64 ビット ARM コントロールプレーンマシンを使用し、すでに 64 ビット ARM コンピュートマシンが含まれているクラスターに 64 ビット x86 コンピュートマシンを追加します。この場合、64 ビット x86 がセカンダリーアーキテクチャーと見なされます。

- 64 ビット x86 コントロールプレーンマシンを使用し、すでに 64 ビット x86 コンピュートマシンが含まれているクラスターに 64 ビット ARM コンピュートマシンを追加します。この場合、64 ビット ARM がセカンダリーアーキテクチャーと見なされます。

注記

セカンダリーアーキテクチャーノードをクラスターに追加する前に、Multiarch Tuning Operator をインストールし、**ClusterPodPlacementConfig** カスタムリソースをデプロイすることを推奨します。詳細は、「Multiarch Tuning Operator を使用してマルチアーキテクチャークラスター上のワークロードを管理する」を参照してください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- インストールプログラムを使用して、マルチアーキテクチャーインストーラーバイナリーを使用する 64 ビット x86 または 64 ビット ARM シングルアーキテクチャー GCP クラスターを作成した。

手順

1. OpenShift CLI (**oc**) にログインします。
2. YAML ファイルを作成し、設定を追加して、クラスター内の 64 ビット ARM または 64 ビット x86 コンピュートノードを制御するコンピュートマシンセットを作成します。

GCP の 64 ビット ARM または 64 ビット x86 コンピュートノードの MachineSet オブジェクトの例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-w-a
  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>-w-a
  template:
    metadata:
      creationTimestamp: null
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: <role> 2
        machine.openshift.io/cluster-api-machine-type: <role>
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-w-a
    spec:
      metadata:
        labels:
          node-role.kubernetes.io/<role>: ""
      providerSpec:
```

```
      value:
        apiVersion: gcpprovider.openshift.io/v1beta1
        canIPForward: false
        credentialsSecret:
          name: gcp-cloud-credentials
        deletionProtection: false
        disks:
        - autoDelete: true
          boot: true
          image: <path_to_image> 3
          labels: null
          sizeGb: 128
          type: pd-ssd
        gcpMetadata: 4
        - key: <custom_metadata_key>
          value: <custom_metadata_value>
        kind: GCPMachineProviderSpec
        machineType: n1-standard-4 5
        metadata:
          creationTimestamp: null
        networkInterfaces:
        - network: <infrastructure_id>-network
          subnetwork: <infrastructure_id>-worker-subnet
        projectID: <project_name> 6
        region: us-central1 7
        serviceAccounts:
        - email: <infrastructure_id>-w@<project_name>.iam.gserviceaccount.com
          scopes:
          - https://www.googleapis.com/auth/cloud-platform
        tags:
          - <infrastructure_id>-worker
        userDataSecret:
          name: worker-user-data
        zone: us-central1-a
```

1. クラスターのプロビジョニング時に設定したクラスター ID を基にするインフラストラクチャー ID を指定します。以下のコマンドを実行してインフラストラクチャー ID を取得できます。

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructure cluster
```

2. 追加するロールノードラベルを指定します。

3. 現在のコンピュートマシンセットで使用されるイメージへのパスを指定します。イメージへのパスにはプロジェクトとイメージ名が必要です。

   プロジェクトとイメージ名にアクセスするには、次のコマンドを実行します。

```
$ oc get configmap/coreos-bootimages \
  -n openshift-machine-config-operator \
  -o jsonpath='{.data.stream}' | jq \
  -r '.architectures.aarch64.images.gcp'
```

出力例

```
  "gcp": {
    "release": "415.92.202309142014-0",
    "project": "rhcos-cloud",
    "name": "rhcos-415-92-202309142014-0-gcp-aarch64"
  }
```

出力の **project** パラメーターと **name** パラメーターを使用して、マシンセット内のイメージフィールドへのパスを作成します。イメージへのパスは次の形式に従う必要があります。

```
$ projects/<project>/global/images/<image_name>
```

4. オプション: **key:value** のペアの形式でカスタムメタデータを指定します。ユースケースの例については、カスタムメタデータの設定 について GCP のドキュメントを参照してください。
5. 選択した OS イメージの CPU アーキテクチャーに合ったマシンタイプを指定します。詳細は、「64 ビット ARM インフラストラクチャー上の GCP のテスト済みのインスタンスタイプ」を参照してください。
6. クラスターに使用する GCP プロジェクトの名前を指定します。
7. リージョンを指定します。たとえば、**us-central1** です。選択したゾーンに必要なアーキテクチャーを備えたマシンがあることを確認してください。

3. 次のコマンドを実行してコンピュートマシンセットを作成します。

```
$ oc create -f <file_name> 1
```

1. **<file_name>** は、コンピュートマシン設定を含む YAML ファイルの名前に置き換えます。たとえば、**gcp-arm64-machine-set-0.yaml**、または **gcp-amd64-machine-set-0.yaml** です。

検証

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

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

出力に、作成したマシンセットが含まれている必要があります。

出力例

```
NAME                                 DESIRED  CURRENT  READY  AVAILABLE  AGE
<infrastructure_id>-gcp-machine-set-0               2       2     2         2  10m
```

2. 次のコマンドを実行すると、ノードが準備完了状態でスケジュール可能かどうかを確認できます。

```
$ oc get nodes
```

関連情報

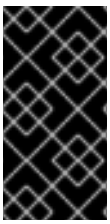
関連情報

- 64 ビット ARM インフラストラクチャー上の GCP のテスト済みインスタンスタイプ
- Multiarch Tuning Operator を使用してマルチアーキテクチャークラスター上のワークロードを管理する

4.5. ベアメタル、IBM POWER、または IBM Z 上でマルチアーキテクチャーのコンピューティングマシンを含むクラスターを作成する

ベアメタル (**x86_64** または **aarch64**)、IBM Power® (**ppc64le**)、または IBM Z® (**s390x**) 上にマルチアーキテクチャーコンピュートマシンを含むクラスターを作成するには、そのプラットフォームに既存のシングルアーキテクチャークラスターが必要です。ご使用のプラットフォームのインストール手順に従ってください。

- ユーザーによってプロビジョニングされるクラスターのベアメタルへのインストールその後、ベアメタル上の OpenShift Container Platform クラスターに 64 ビット ARM コンピューティングマシンを追加できます。
- クラスターの IBM Power® へのインストール その後、IBM Power® 上の OpenShift Container Platform クラスターに **x86_64** コンピュートマシンを追加できます。
- クラスターの IBM Z® および IBM® LinuxONE へのインストール その後、IBM Z® および IBM® LinuxONE 上の OpenShift Container Platform クラスターに **x86_64** コンピュートマシンを追加できます。



重要

ベアメタルの installer-provisioned infrastructure および Bare Metal Operator は、クラスターの初期セットアップ中にセカンダリーアーキテクチャーノードを追加することをサポートしていません。セカンダリーアーキテクチャーノードは、初期クラスターのセットアップ後にのみ手動で追加できます。

クラスターに追加のコンピュートノードを追加する前に、クラスターをマルチアーキテクチャーペイロードを使用するクラスターにアップグレードする必要があります。マルチアーキテクチャーペイロード への移行の詳細は、マルチアーキテクチャーコンピューティングマシンを使用したクラスターへ の移行 を参照してください。

次の手順では、ISO イメージまたはネットワーク PXE ブートを使用して RHCOS コンピューティングマシンを作成する方法について説明します。これにより、クラスターにノードを追加し、マルチアーキテクチャーコンピュートマシンを含むクラスターをデプロイできるようになります。

注記

セカンダリーアーキテクチャーノードをクラスターに追加する前に、Multiarch Tuning Operator をインストールし、**ClusterPodPlacementConfig** オブジェクトをデプロイすることを推奨します。詳細は、Multiarch Tuning Operator を使用してマルチアーキテクチャークラスター上のワークロードを管理する を参照してください。

4.5.1. クラスターの互換性の確認

異なるアーキテクチャーのコンピュートノードをクラスターに追加する前に、クラスターがマルチアーキテクチャー互換であることを確認する必要があります。

前提条件

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

手順

1. OpenShift CLI (**oc**) にログインします。
2. 次のコマンドを実行すると、クラスターがアーキテクチャーペイロードを使用していることを確認できます。

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

検証

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用しています。

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

その後、クラスターへのマルチアーキテクチャーコンピュートノードの追加を開始できます。

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用していません。

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

重要

クラスターがマルチアーキテクチャーコンピューティングマシンをサポートするようにクラスターを移行するには、マルチアーキテクチャーコンピューティングマシンを 使用したクラスターへ の移行 の手順に従います。

4.5.2. ISO イメージを使用した 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) コンピュートマシンを作成できます。

前提条件

- クラスターのコンピュートマシンの Ignition 設定ファイルの URL を取得します。このファイルがインストール時に HTTP サーバーにアップロードされている必要があります。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 次のコマンドを実行して、クラスターから Ignition 設定ファイルを抽出します。

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --
to=- > worker.ign
```

2. クラスターからエクスポートした **worker.ign** Ignition 設定ファイルを HTTP サーバーにアップロードします。これらのファイルの URL をメモします。

3. Ignition ファイルが URL で利用可能であることを検証できます。次の例では、コンピュートノードの Ignition 設定ファイルを取得します。

```
$ curl -k http://<HTTP_server>/worker.ign
```

4. 次のコマンドを実行すると、新しいマシンを起動するための ISO イメージにアクセスできます。

```
RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get
configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.
<architecture>.artifacts.metal.formats.iso.disk.location')
```

5. ISO ファイルを使用して、追加のコンピュートマシンに RHCOS をインストールします。クラスターのインストール前にマシンを作成する際に使用したのと同じ方法を使用します。

   - ディスクに ISO イメージを書き込み、これを直接起動します。
   - LOM インターフェイスで ISO リダイレクトを使用します。

6. 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

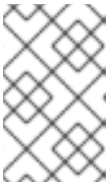
RHCOS インストールの起動プロセスを中断して、カーネル引数を追加できます。ただし、この ISO 手順では、カーネル引数を追加する代わりに、次の手順で概説するように **coreos-installer** コマンドを使用する必要があります。

7. **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> 1 2
```

1. **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。

2. **--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場合は、**coreos-installer** を実行する前に、内部認証局 (CA) をシステムのトラスト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
```

8. 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。

9. 継続してクラスター用の追加のコンピュートマシンを作成します。

4.5.3. PXE または iPXE ブートによる RHCOS マシンの作成

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

前提条件

- クラスターのコンピュートマシンの Ignition 設定ファイルの URL を取得します。このファイルがインストール時に HTTP サーバーにアップロードされている必要があります。
- クラスターのインストール時に HTTP サーバーにアップロードした RHCOS ISO イメージ、圧縮されたメタル BIOS、**kernel**、および **initramfs** ファイルの URL を取得します。
- インストール時に OpenShift Container Platform クラスターのマシンを作成するために使用した PXE ブートインフラストラクチャーにアクセスできる必要があります。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> (1)
    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 (2)
```

(1) 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。

(2) 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。

注記

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

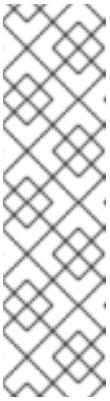
- iPXE (**x86_**64 + **aarch64**):

```
kernel http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> initrd=main
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign (1) (2)
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img (3)
boot
```

(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.live.rootfs_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.inst.ignition_url** パラメーター値は **rootfs** のライブファイルの場所です。

(2) 複数の NIC を使用する場合、**ip** オプションに単一インターフェイスを指定します。たとえば、**eno1** という名前の NIC で DHCP を使用するには、**ip=eno1:dhcp** を設定します。

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



注記

この設定では、グラフィカルコンソールを備えたマシンでのシリアルコンソールアクセスは有効になりません。別のコンソールを設定するには、**kernel** 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? と、「高度な RHCOS インストール設定」セクションの「PXE および ISO インストール用シリアルコンソールの有効化」を参照してください。

注記

aarch64 アーキテクチャーで CoreOS **kernel** をネットワークブートするには、**IMAGE_GZIP** オプションが有効になっているバージョンの iPXE ビルドを使用する必要があります。iPXE の **IMAGE_GZIP** オプション を参照してください。

- **aarch64** 上の PXE (第 2 段階として UEFI および GRUB を使用) の場合:

```
menuentry 'Install CoreOS' {
    linux rhcos-<version>-live-kernel-<architecture>
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign (1) (2)
    initrd rhcos-<version>-live-initramfs.<architecture>.img (3)
}
```

(1) HTTP/TFTP 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、TFTP サーバー上の **kernel** ファイルの場所になります。**coreos.live.rootfs_url** パラメーター値は **rootfs** ファイルの場所であり、**coreos.inst.ignition_url** パラメーター値は HTTP サーバー上のブートストラップ Ignition 設定ファイルの場所になります。

(2) 複数の NIC を使用する場合、**ip** オプションに単一インターフェイスを指定します。たとえば、**eno1** という名前の NIC で DHCP を使用するには、**ip=eno1:dhcp** を設定します。

(3) TFTP 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。

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

4.5.4. 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  v1.29.4
master-1  Ready     master  63m  v1.29.4
master-2  Ready     master  64m  v1.29.4
```

出力には作成したすべてのマシンがリスト表示されます。

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。

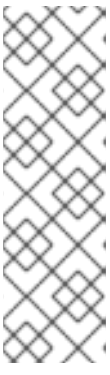
```
$ oc get csr
```

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
...
```

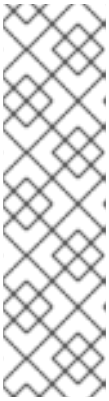
この例では、2 つのマシンがクラスターに参加しています。このリストにはさらに多くの承認された CSR が表示される可能性があります。

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 user-provisioned infrastructure 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 のリストからの 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 のリストからの 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  v1.29.4
master-1  Ready     master  73m  v1.29.4
master-2  Ready     master  74m  v1.29.4
worker-0  Ready     worker  11m  v1.29.4
worker-1  Ready     worker  11m  v1.29.4
```

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。

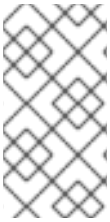
4.6. Z/VM を使用した IBM Z および IBM LINUXONE 上でマルチアーキテクチャーのコンピューティングマシンを含むクラスターを作成する

z/VM を使用して IBM Z® および IBM® LinuxONE (**s390x**) 上にマルチアーキテクチャーのコンピュートマシンを含むクラスターを作成するには、既存の単一アーキテクチャーの **x86_64** クラスターが必要です。その後、**s390x** コンピュートマシンを OpenShift Container Platform クラスターに追加できます。

s390x ノードをクラスターに追加する前に、クラスターをマルチアーキテクチャーペイロードを使用するクラスターにアップグレードする必要があります。マルチアーキテクチャーペイロード への移行の詳細は、マルチアーキテクチャーコンピューティングマシンを使用したクラスターへ の移行 を参照してください。

次の手順では、z/VM インスタンスを使用して RHCOS コンピュートマシンを作成する方法を説明します。これにより、**s390x** ノードをクラスターに追加し、マルチアーキテクチャーのコンピュートマシンを含むクラスターをデプロイメントできるようになります。

x **86_64 でマルチアーキテクチャーコンピュートマシンを使用する IBM Z® または IBM® LinuxONE (s390x**)クラスターを作成するには、IBM Z® および IBM® LinuxONE へのクラスターのインストール の手順に従います。その後、ベアメタル、IBM Power、または IBM Z 上でマルチアーキテクチャーコンピュートマシンを含むクラスターを作成する の説明に従って、**x86_64** コンピュートマシンを追加できます。



注記

セカンダリーアーキテクチャーノードをクラスターに追加する前に、Multiarch Tuning Operator をインストールし、**ClusterPodPlacementConfig** オブジェクトをデプロイすることを推奨します。詳細は、Multiarch Tuning Operator を使用してマルチアーキテクチャークラスター上のワークロードを管理する を参照してください。

4.6.1. クラスターの互換性の確認

異なるアーキテクチャーのコンピュートノードをクラスターに追加する前に、クラスターがマルチアーキテクチャー互換であることを確認する必要があります。

前提条件

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

手順

1. OpenShift CLI (**oc**) にログインします。
2. 次のコマンドを実行すると、クラスターがアーキテクチャーペイロードを使用していることを確認できます。

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

検証

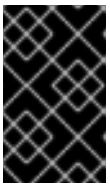
- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用しています。

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

その後、クラスターへのマルチアーキテクチャーコンピュートノードの追加を開始できます。

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用していません。

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```



重要

クラスターがマルチアーキテクチャーコンピューティングマシンをサポートするようにクラスターを移行するには、マルチアーキテクチャーコンピューティングマシンを 使用したクラスターへ の移行 の手順に従います。

4.6.2. z/VM を使用した IBM Z 上での RHCOS マシンの作成

z/VM を使用して IBM Z® 上で実行する Red Hat Enterprise Linux CoreOS (RHCOS) コンピュートマシンをさらに作成し、既存のクラスターに接続できます。

前提条件

- ノードのホスト名および逆引き参照を実行できるドメインネームサーバー (DNS) がある。
- 作成するマシンがアクセスできるプロビジョニングマシンで稼働している HTTP または HTTPS サーバーがある。

手順

1. UDP アグリゲーションを無効にします。
   現在、UDP アグリゲーションは IBM Z® ではサポートされておらず、**x86_64** コントロールプレーンと追加の **s390x** コンピュートマシンを備えたマルチアーキテクチャーコンピュートクラスターでは自動的に非アクティブ化されません。追加のコンピュートノードがクラスターに正しく追加されるようにするには、UDP アグリゲーションを手動で無効にする必要があります。

   a. 次の内容を含む YAML ファイル **udp-aggregation-config.yaml** を作成します。

   ```
   apiVersion: v1
   kind: ConfigMap
   data:
     disable-udp-aggregation: "true"
   metadata:
     name: udp-aggregation-config
     namespace: openshift-network-operator
   ```

   b. 次のコマンドを実行して、ConfigMap リソースを作成します。

   ```
   $ oc create -f udp-aggregation-config.yaml
   ```

2. 次のコマンドを実行して、クラスターから Ignition 設定ファイルを抽出します。

   ```
   $ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --
   to=- > worker.ign
   ```

3. クラスターからエクスポートした **worker.ign** Ignition 設定ファイルを HTTP サーバーにアップロードします。このファイルの URL をメモします。

4. Ignition ファイルが URL で利用可能であることを検証できます。次の例では、コンピュートノードの Ignition 設定ファイルを取得します。

   ```
   $ curl -k http://<http_server>/worker.ign
   ```

5. 次のコマンドを実行して、RHEL ライブ **kernel**、**initramfs**、および **rootfs** ファイルをダウンロードします。

   ```
   $ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
   jsonpath='{.data.stream}' \
   | jq -r '.architectures.s390x.artifacts.metal.formats.pxe.kernel.location')
   ```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.initramfs.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.rootfs.location')
```

6. ダウンロードした RHEL ライブ **kernel**、**initramfs**、および **rootfs** ファイルを、追加する z/VM ゲストからアクセスできる HTTP または HTTPS サーバーに移動します。

7. z/VM ゲストのパラメーターファイルを作成します。次のパラメーターは仮想マシンに固有です。

   - オプション: 静的 IP アドレスを指定するには、次のエントリーをコロンで区切って **ip=** パラメーターを追加します。

     i. マシンの IP アドレス。

     ii. 空の文字列。

     iii. ゲートウェイ。

     iv. ネットマスク。

     v. **hostname.domainname** 形式のマシンホストおよびドメイン名。この値を省略して、RHCOS に決定させるようにします。

     vi. ネットワークインターフェイス名。この値を省略して、RHCOS に決定させるようにします。

     vii. 値 **none**。

   - **coreos.inst.ignition_url=** には、**worker.ign** ファイルへの URL を指定します。HTTP プロトコルおよび HTTPS プロトコルのみがサポートされます。

   - **coreos.live.rootfs_url=** の場合、起動している **kernel** および **initramfs** の一致する rootfs アーティファクトを指定します。HTTP プロトコルおよび HTTPS プロトコルのみがサポートされます。

   - DASD タイプのディスクへのインストールには、以下のタスクを実行します。

     i. **coreos.inst.install_dev=** には、**/dev/dasda** を指定します。

     ii. **rd.dasd=** を使用して、RHCOS がインストールされる DASD を指定します。

     iii. 必要に応じてさらにパラメーターを調整できます。
     以下はパラメーターファイルの例、**additional-worker-dasd.parm** です。

```
rd.neednet=1 \
console=ttysclp0 \
coreos.inst.install_dev=/dev/dasda \
coreos.live.rootfs_url=http://<http_server>/rhcos-<version>-live-rootfs.
<architecture>.img \
coreos.inst.ignition_url=http://<http_server>/worker.ign \
ip=<ip>::<gateway>:<netmask>:<hostname>::none nameserver=<dns> \
```

```
rd.znet=qeth,0.0.bdf0,0.0.bdf1,0.0.bdf2,layer2=1,portno=0 \
zfcp.allow_lun_scan=0 \
rd.dasd=0.0.3490
```

パラメーターファイルのすべてのオプションを 1 行で記述し、改行文字がないことを確認します。

- FCP タイプのディスクへのインストールには、以下のタスクを実行します。

  i. **rd.zfcp=<adapter>,<wwpn>,<lun>** を使用して RHCOS がインストールされる FCP ディスクを指定します。マルチパスの場合、それぞれの追加のステップについてこのステップを繰り返します。

  注記

  複数のパスを使用してインストールする場合は、問題が発生する可能性があるため、後でではなくインストールの直後にマルチパスを有効にする必要があります。

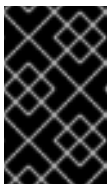
  ii. インストールデバイスを **coreos.inst.install_dev=/dev/sda** として設定します。

  注記

  追加の LUN が NPIV で設定される場合は、FCP に **zfcp.allow_lun_scan=0** が必要です。CSI ドライバーを使用するために **zfcp.allow_lun_scan=1** を有効にする必要がある場合などには、各ノードが別のノードのブートパーティションにアクセスできないように NPIV を設定する必要があります。

  iii. 必要に応じてさらにパラメーターを調整できます。

  重要

  マルチパスを完全に有効にするには、インストール後の追加の手順が必要です。詳細は、**インストール後のマシン設定タスク** 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。

  以下は、マルチパスを使用するワーカーノードのパラメーターファイルの例 **additional-worker-fcp.parm** です。

```
rd.neednet=1 \
console=ttysclp0 \
coreos.inst.install_dev=/dev/sda \
coreos.live.rootfs_url=http://<http_server>/rhcos-<version>-live-rootfs.
<architecture>.img \
coreos.inst.ignition_url=http://<http_server>/worker.ign \
ip=<ip>::<gateway>:<netmask>:<hostname>::none nameserver=<dns> \
rd.znet=qeth,0.0.bdf0,0.0.bdf1,0.0.bdf2,layer2=1,portno=0 \
zfcp.allow_lun_scan=0 \
rd.zfcp=0.0.1987,0x50050763070bc5e3,0x4008400B00000000 \
rd.zfcp=0.0.19C7,0x50050763070bc5e3,0x4008400B00000000 \
rd.zfcp=0.0.1987,0x50050763071bc5e3,0x4008400B00000000 \
rd.zfcp=0.0.19C7,0x50050763071bc5e3,0x4008400B00000000
```

パラメーターファイルのすべてのオプションを 1 行で記述し、改行文字がないことを確認します。

8. FTP などを使用し、**initramfs**、**kernel**、パラメーターファイル、および RHCOS イメージを z/VM に転送します。FTP でファイルを転送し、仮想リーダーから起動する方法については、Z/VM 環境へのインストール を参照してください。

9. ファイルを z/VM ゲスト仮想マシンの仮想リーダーに punch します。
IBM® ドキュメントの PUNCH を参照してください。

ヒント

CP PUNCH コマンドを使用するか、Linux を使用している場合は、**vmur** コマンドを使用して 2 つの z/VM ゲスト仮想マシン間でファイルを転送できます。

10. ブートストラップマシンで CMS にログインします。

11. 次のコマンドを実行して、リーダーからブートストラップマシンを IPL します。

```
$ ipl c
```

IBM® ドキュメントの IPL を参照してください。

4.6.3. 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

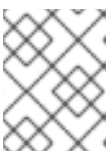
1. クラスターがマシンを認識していることを確認します。

```
$ oc get nodes
```

出力例

```
NAME     STATUS    ROLES   AGE  VERSION
master-0  Ready     master  63m  v1.29.4
master-1  Ready     master  63m  v1.29.4
master-2  Ready     master  64m  v1.29.4
```

出力には作成したすべてのマシンがリスト表示されます。



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。このリスト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 user-provisioned infrastructure 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。

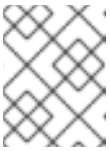
- それらを個別に承認するには、それぞれの有効な CSR について以下のコマンドを実行します。

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** は、現行の CSR のリストからの 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 のリストからの 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  v1.29.4
master-1  Ready     master  73m  v1.29.4
master-2  Ready     master  74m  v1.29.4
worker-0  Ready     worker  11m  v1.29.4
worker-1  Ready     worker  11m  v1.29.4
```

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。

4.7. IBM Z および IBM LINUXONE 上の LPAR にマルチアーキテクチャーコンピュートマシンを含むクラスターを作成する

IBM Z® および IBM® LinuxONE (**s390x**) 上の LPAR にマルチアーキテクチャーコンピュートマシンを含むクラスターを作成するには、既存のシングルアーキテクチャーの **x86_64** クラスターが必要です。その後、**s390x** コンピュートマシンを OpenShift Container Platform クラスターに追加できます。

s390x ノードをクラスターに追加する前に、クラスターをマルチアーキテクチャーペイロードを使用するクラスターにアップグレードする必要があります。マルチアーキテクチャーペイロード への移行の詳細は、マルチアーキテクチャーコンピューティングマシンを使用したクラスターへ の移行 を参照してください。

以下の手順では、LPAR インスタンスを使用して RHCOS コンピュートマシンを作成する方法について説明します。これにより、**s390x** ノードをクラスターに追加し、マルチアーキテクチャーのコンピュートマシンを含むクラスターをデプロイメントできるようになります。

注記

x **86_64 でマルチアーキテクチャーコンピュートマシンを使用する IBM Z® または IBM® LinuxONE (s390x**)クラスターを作成するには、IBM Z® および IBM® LinuxONE へのクラスターのインストール の手順に従います。その後、ベアメタル、IBM Power、または IBM Z 上でマルチアーキテクチャーコンピュートマシンを含むクラスターを作成する の説明に従って、**x86_64** コンピュートマシンを追加できます。

4.7.1. クラスターの互換性の確認

異なるアーキテクチャーのコンピュートノードをクラスターに追加する前に、クラスターがマルチアーキテクチャー互換であることを確認する必要があります。

前提条件

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

手順

1. OpenShift CLI (**oc**) にログインします。
2. 次のコマンドを実行すると、クラスターがアーキテクチャーペイロードを使用していることを確認できます。

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

検証

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用しています。

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

その後、クラスターへのマルチアーキテクチャーコンピュートノードの追加を開始できます。

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用していません。

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

重要

クラスターがマルチアーキテクチャーコンピューティングマシンをサポートするようにクラスターを移行するには、マルチアーキテクチャーコンピューティングマシンを 使用したクラスターへ の移行 の手順に従います。

4.7.2. z/VM を使用した IBM Z 上での RHCOS マシンの作成

z/VM を使用して IBM Z® 上で実行する Red Hat Enterprise Linux CoreOS (RHCOS) コンピュートマシンをさらに作成し、既存のクラスターに接続できます。

前提条件

- ノードのホスト名および逆引き参照を実行できるドメインネームサーバー (DNS) がある。
- 作成するマシンがアクセスできるプロビジョニングマシンで稼働している HTTP または HTTPS サーバーがある。

手順

1. UDP アグリゲーションを無効にします。
現在、UDP アグリゲーションは IBM Z® ではサポートされておらず、**x86_64** コントロールプレーンと追加の **s390x** コンピュートマシンを備えたマルチアーキテクチャーコンピュートクラスターでは自動的に非アクティブ化されません。追加のコンピュートノードがクラスターに正しく追加されるようにするには、UDP アグリゲーションを手動で無効にする必要があります。

   a. 次の内容を含む YAML ファイル **udp-aggregation-config.yaml** を作成します。

```
apiVersion: v1
kind: ConfigMap
data:
  disable-udp-aggregation: "true"
metadata:
  name: udp-aggregation-config
  namespace: openshift-network-operator
```

   b. 次のコマンドを実行して、ConfigMap リソースを作成します。

```
$ oc create -f udp-aggregation-config.yaml
```

2. 次のコマンドを実行して、クラスターから Ignition 設定ファイルを抽出します。

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --
to=- > worker.ign
```

3. クラスターからエクスポートした **worker.ign** Ignition 設定ファイルを HTTP サーバーにアップロードします。このファイルの URL をメモします。

4. Ignition ファイルが URL で利用可能であることを検証できます。次の例では、コンピュートノードの Ignition 設定ファイルを取得します。

```
$ curl -k http://<http_server>/worker.ign
```

5. 次のコマンドを実行して、RHEL ライブ **kernel**、**initramfs**、および **rootfs** ファイルをダウンロードします。

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.kernel.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.initramfs.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.rootfs.location')
```

6. ダウンロードした RHEL ライブ **kernel**、**initramfs**、および **rootfs** ファイルを、追加する z/VM ゲストからアクセスできる HTTP または HTTPS サーバーに移動します。

7. z/VM ゲストのパラメーターファイルを作成します。次のパラメーターは仮想マシンに固有です。

   - オプション: 静的 IP アドレスを指定するには、次のエントリーをコロンで区切って **ip=** パラメーターを追加します。

     i. マシンの IP アドレス。

     ii. 空の文字列。

     iii. ゲートウェイ。

     iv. ネットマスク。

     v. **hostname.domainname** 形式のマシンホストおよびドメイン名。この値を省略して、RHCOS に決定させるようにします。

     vi. ネットワークインターフェイス名。この値を省略して、RHCOS に決定させるようにします。

     vii. 値 **none**。

- **coreos.inst.ignition_url=** には、**worker.ign** ファイルへの URL を指定します。HTTP プロトコルおよび HTTPS プロトコルのみがサポートされます。

- **coreos.live.rootfs_url=** の場合、起動している **kernel** および **initramfs** の一致する rootfs アーティファクトを指定します。HTTP プロトコルおよび HTTPS プロトコルのみがサポートされます。

- DASD タイプのディスクへのインストールには、以下のタスクを実行します。

  i. **coreos.inst.install_dev=** には、**/dev/dasda** を指定します。

  ii. **rd.dasd=** を使用して、RHCOS がインストールされる DASD を指定します。

  iii. 必要に応じてさらにパラメーターを調整できます。
  以下はパラメーターファイルの例、**additional-worker-dasd.parm** です。

  ```
  rd.neednet=1 \
  console=ttysclp0 \
  coreos.inst.install_dev=/dev/dasda \
  coreos.live.rootfs_url=http://<http_server>/rhcos-<version>-live-rootfs.
  <architecture>.img \
  coreos.inst.ignition_url=http://<http_server>/worker.ign \
  ip=<ip>::<gateway>:<netmask>:<hostname>::none nameserver=<dns> \
  rd.znet=qeth,0.0.bdf0,0.0.bdf1,0.0.bdf2,layer2=1,portno=0 \
  zfcp.allow_lun_scan=0 \
  rd.dasd=0.0.3490
  ```

  パラメーターファイルのすべてのオプションを 1 行で記述し、改行文字がないことを確認します。

- FCP タイプのディスクへのインストールには、以下のタスクを実行します。

  i. **rd.zfcp=<adapter>,<wwpn>,<lun>** を使用して RHCOS がインストールされる FCP ディスクを指定します。マルチパスの場合、それぞれの追加のステップについてこのステップを繰り返します。

  注記

  複数のパスを使用してインストールする場合は、問題が発生する可能性があるため、後でではなくインストールの直後にマルチパスを有効にする必要があります。

  ii. インストールデバイスを **coreos.inst.install_dev=/dev/sda** として設定します。

  注記

  追加の LUN が NPIV で設定される場合は、FCP に **zfcp.allow_lun_scan=0** が必要です。CSI ドライバーを使用するために **zfcp.allow_lun_scan=1** を有効にする必要がある場合などには、各ノードが別のノードのブートパーティションにアクセスできないように NPIV を設定する必要があります。

  iii. 必要に応じてさらにパラメーターを調整できます。

重要

マルチパスを完全に有効にするには、インストール後の追加の手順が必要です。詳細は、**インストール後のマシン設定タスク** 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。

以下は、マルチパスを使用するワーカーノードのパラメーターファイルの例 **additional-worker-fcp.parm** です。

```
rd.neednet=1 \
console=ttysclp0 \
coreos.inst.install_dev=/dev/sda \
coreos.live.rootfs_url=http://<http_server>/rhcos-<version>-live-rootfs.
<architecture>.img \
coreos.inst.ignition_url=http://<http_server>/worker.ign \
ip=<ip>::<gateway>:<netmask>:<hostname>::none nameserver=<dns> \
rd.znet=qeth,0.0.bdf0,0.0.bdf1,0.0.bdf2,layer2=1,portno=0 \
zfcp.allow_lun_scan=0 \
rd.zfcp=0.0.1987,0x50050763070bc5e3,0x4008400B00000000 \
rd.zfcp=0.0.19C7,0x50050763070bc5e3,0x4008400B00000000 \
rd.zfcp=0.0.1987,0x50050763071bc5e3,0x4008400B00000000 \
rd.zfcp=0.0.19C7,0x50050763071bc5e3,0x4008400B00000000
```

パラメーターファイルのすべてのオプションを 1 行で記述し、改行文字がないことを確認します。

8. FTP などを使用し、**initramfs**、**kernel**、パラメーターファイル、および RHCOS イメージを z/VM に転送します。FTP でファイルを転送し、仮想リーダーから起動する方法については、Z/VM 環境へのインストール を参照してください。

9. ファイルを z/VM ゲスト仮想マシンの仮想リーダーに punch します。IBM® ドキュメントの PUNCH を参照してください。

   ヒント

   CP PUNCH コマンドを使用するか、Linux を使用している場合は、**vmur** コマンドを使用して 2 つの z/VM ゲスト仮想マシン間でファイルを転送できます。

10. ブートストラップマシンで CMS にログインします。

11. 次のコマンドを実行して、リーダーからブートストラップマシンを IPL します。

    ```
    $ ipl c
    ```

    IBM® ドキュメントの IPL を参照してください。

4.7.3. 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  v1.29.4
master-1  Ready     master  63m  v1.29.4
master-2  Ready     master  64m  v1.29.4
```

出力には作成したすべてのマシンがリスト表示されます。

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

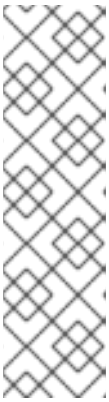
```
NAME        AGE     REQUESTOR                                                                   CONDITION
csr-8b2br   15m     system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
csr-8vnps   15m     system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   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。このリスト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 user-provisioned infrastructure 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。

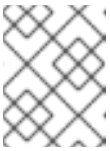
- それらを個別に承認するには、それぞれの有効な CSR について以下のコマンドを実行します。

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** は、現行の CSR のリストからの 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 のリストからの 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。

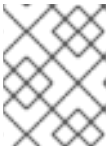
```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs oc adm certificate approve
```

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  v1.29.4
master-1  Ready     master  73m  v1.29.4
master-2  Ready     master  74m  v1.29.4
worker-0  Ready     worker  11m  v1.29.4
worker-1  Ready     worker  11m  v1.29.4
```



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。

4.8. RHEL KVM を使用した IBM Z および IBM LINUXONE 上でマルチアーキテクチャーのコンピュートマシンを含むクラスターを作成する

RHEL KVM を使用して IBM Z® および IBM® LinuxONE (**s390x**) 上のマルチアーキテクチャーコンピュートマシンでクラスターを作成するには、既存の単一アーキテクチャー **x86_64** クラスターが必要です。その後、**s390x** コンピュートマシンを OpenShift Container Platform クラスターに追加できます。

s390x ノードをクラスターに追加する前に、クラスターをマルチアーキテクチャーペイロードを使用するクラスターにアップグレードする必要があります。マルチアーキテクチャーペイロード への移行の詳細は、マルチアーキテクチャーコンピューティングマシンを使用したクラスターへ の移行 を参照してください。

次の手順では、RHEL KVM インスタンスを使用して RHCOS コンピュートマシンを作成する方法を説明します。これにより、**s390x** ノードをクラスターに追加し、マルチアーキテクチャーのコンピュートマシンを含むクラスターをデプロイメントできるようになります。

x **86_64 でマルチアーキテクチャーコンピュートマシンを使用する IBM Z® または IBM® LinuxONE (s390x**)クラスターを作成するには、IBM Z® および IBM® LinuxONE へのクラスターのインストール の手順に従います。その後、ベアメタル、IBM Power、または IBM Z 上でマルチアーキテクチャーコンピュートマシンを含むクラスターを作成する の説明に従って、**x86_64** コンピュートマシンを追加できます。

注記

セカンダリーアーキテクチャーノードをクラスターに追加する前に、Multiarch Tuning Operator をインストールし、**ClusterPodPlacementConfig** オブジェクトをデプロイすることを推奨します。詳細は、Multiarch Tuning Operator を使用してマルチアーキテクチャークラスター上のワークロードを管理する を参照してください。

4.8.1. クラスターの互換性の確認

異なるアーキテクチャーのコンピュートノードをクラスターに追加する前に、クラスターがマルチアーキテクチャー互換であることを確認する必要があります。

前提条件

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

手順

1. OpenShift CLI (**oc**) にログインします。
2. 次のコマンドを実行すると、クラスターがアーキテクチャーペイロードを使用していることを確認できます。

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

検証

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用しています。

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

その後、クラスターへのマルチアーキテクチャーコンピュートノードの追加を開始できます。

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用していません。

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

重要

クラスターがマルチアーキテクチャーコンピューティングマシンをサポートするようにクラスターを移行するには、マルチアーキテクチャーコンピューティングマシンを 使用したクラスターへ の移行 の手順に従います。

4.8.2. virt-install を使用した RHCOS マシンの作成

virt-install を使用すると、クラスター用にさらに Red Hat Enterprise Linux CoreOS (RHCOS) コンピュートマシンを作成できます。

前提条件

- この手順では RHEL KVM ホストと呼ばれる、KVM を使用する RHEL 8.7 以降で実行されている少なくとも 1 つの LPAR がある。
- KVM/QEMU ハイパーバイザーが RHEL KVM ホストにインストーされている
- ノードのホスト名および逆引き参照を実行できるドメインネームサーバー (DNS) がある。
- HTTP または HTTPS サーバーが設定されている。

手順

1. UDP アグリゲーションを無効にします。
   現在、UDP アグリゲーションは IBM Z® ではサポートされておらず、**x86_64** コントロールプレーンと追加の **s390x** コンピュートマシンを備えたマルチアーキテクチャーコンピュートクラスターでは自動的に非アクティブ化されません。追加のコンピュートノードがクラスターに正しく追加されるようにするには、UDP アグリゲーションを手動で無効にする必要があります。

   a. 次の内容を含む YAML ファイル **udp-aggregation-config.yaml** を作成します。

   ```
   apiVersion: v1
   kind: ConfigMap
   data:
     disable-udp-aggregation: "true"
   metadata:
     name: udp-aggregation-config
     namespace: openshift-network-operator
   ```

   b. 次のコマンドを実行して、ConfigMap リソースを作成します。

   ```
   $ oc create -f udp-aggregation-config.yaml
   ```

2. 次のコマンドを実行して、クラスターから Ignition 設定ファイルを抽出します。

   ```
   $ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --to=- > worker.ign
   ```

3. クラスターからエクスポートした **worker.ign** Ignition 設定ファイルを HTTP サーバーにアップロードします。このファイルの URL をメモします。

4. Ignition ファイルが URL で利用可能であることを検証できます。次の例では、コンピュートノードの Ignition 設定ファイルを取得します。

   ```
   $ curl -k http://<HTTP_server>/worker.ign
   ```

5. 次のコマンドを実行して、RHEL ライブ **kernel**、**initramfs**、および **rootfs** ファイルをダウンロードします。

```
 $ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.kernel.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.initramfs.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o
jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.rootfs.location')
```

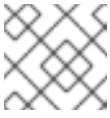
6. **virt-install** を起動する前に、ダウンロードした RHEL ライブの **kernel** ファイル、**initramfs** ファイル、および **rootfs** ファイルを HTTP または HTTPS サーバーに移動します。

7. RHEL **kernel**、**initramfs**、および Ignition ファイル、新規ディスクイメージ、および調整された parm 引数を使用して、新規 KVM ゲストノードを作成します。

```
$ virt-install \
  --connect qemu:///system \
  --name <vm_name> \
  --autostart \
  --os-variant rhel9.4 \ 1
  --cpu host \
  --vcpus <vcpus> \
  --memory <memory_mb> \
  --disk <vm_name>.qcow2,size=<image_size> \
  --network network=<virt_network_parm> \
  --location <media_location>,kernel=<rhcos_kernel>,initrd=<rhcos_initrd> \ 2
  --extra-args "rd.neednet=1" \
  --extra-args "coreos.inst.install_dev=/dev/vda" \
  --extra-args "coreos.inst.ignition_url=http://<http_server>/worker.ign " \ 3
  --extra-args "coreos.live.rootfs_url=http://<http_server>/rhcos-<version>-live-rootfs.
<architecture>.img" \ 4
  --extra-args "ip=<ip>::<gateway>:<netmask>:<hostname>::none" \ 5
  --extra-args "nameserver=<dns>" \
  --extra-args "console=ttysclp0" \
  --noautoconsole \
  --wait
```

1. **os-variant** には、RHCOS コンピュートマシンの RHEL バージョンを指定します。**rhel9.4** が推奨バージョンです。オペレーティングシステムのサポートされている RHEL バージョンを照会するには、次のコマンドを実行します。

   ```
   $ osinfo-query os -f short-id
   ```

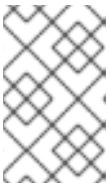
   

   注記

   os-variant では大文字と小文字が区別されます。

2. **--location** には、HTTP サーバーまたは HTTPS サーバーのカーネル/initrd の場所を指定します。

3. **worker.ign** 設定ファイルの場所を指定します。HTTP プロトコルおよび HTTPS プロトコルのみがサポートされます。
4. 起動する **kernel** と **initramfs** の **rootfs** アーティファクトの場所を指定します。HTTP および HTTPS プロトコルのみがサポートされています。
5. オプション: **hostname** には、クライアントマシンの完全修飾ホスト名を指定します。



注記

HAProxy をロードバランサーとして使用している場合は、**/etc/haproxy/haproxy.cfg** 設定ファイル内の **ingress-router-443** および **ingress-router-80** の HAProxy ルールを更新します。

8. 継続してクラスター用の追加のコンピュートマシンを作成します。

4.8.3. 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  v1.29.4
master-1 Ready    master  63m  v1.29.4
master-2 Ready    master  64m  v1.29.4
```

出力には作成したすべてのマシンがリスト表示されます。

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。このリスト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 user-provisioned infrastructure 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 のリストからの 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 のリストからの 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  v1.29.4
master-1  Ready     master  73m  v1.29.4
master-2  Ready     master  74m  v1.29.4
worker-0  Ready     worker  11m  v1.29.4
worker-1  Ready     worker  11m  v1.29.4
```

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。

4.9. IBM POWER 上でマルチアーキテクチャーのコンピュートマシンを含むクラスターを作成する

IBM Power® (**ppc64le**) 上でマルチアーキテクチャーのコンピュートマシンを含むクラスターを作成するには、既存の単一アーキテクチャー (**x86_64**) クラスターが必要です。その後、**ppc64le** コンピュートマシンを OpenShift Container Platform クラスターに追加できます。

重要

ppc64le ノードをクラスターに追加する前に、クラスターをマルチアーキテクチャーペイロードを使用するクラスターにアップグレードする必要があります。マルチアーキテクチャーペイロード への移行の詳細は、マルチアーキテクチャーコンピューティングマシンを使用したクラスターへ の移行 を参照してください。

次の手順では、ISO イメージまたはネットワーク PXE ブートを使用して RHCOS コンピューティングマシンを作成する方法について説明します。これにより、**ppc64le** ノードをクラスターに追加し、マルチアーキテクチャーのコンピュートマシンを含むクラスターをデプロイできるようになります。

x86_64 にマルチアーキテクチャーのコンピューティングマシンを含む IBM Power®(**ppc64le**)クラスターを作成するには、IBM Power® にクラスターをインストール する 手順に従います。その後、ベアメタル、IBM Power、または IBM Z 上でマルチアーキテクチャーコンピュートマシンを含むクラスターを作成する の説明に従って、**x86_64** コンピュートマシンを追加できます。

注記

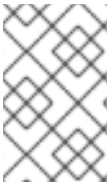
セカンダリーアーキテクチャーノードをクラスターに追加する前に、Multiarch Tuning Operator をインストールし、**ClusterPodPlacementConfig** オブジェクトをデプロイすることを推奨します。詳細は、Multiarch Tuning Operator を使用してマルチアーキテクチャークラスター上のワークロードを管理する を参照してください。

4.9.1. クラスターの互換性の確認

異なるアーキテクチャーのコンピュートノードをクラスターに追加する前に、クラスターがマルチアーキテクチャー互換であることを確認する必要があります。

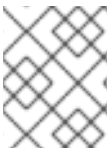
前提条件

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



注記

複数のアーキテクチャーを使用する場合、OpenShift Container Platform ノードのホストは同じストレージレイヤーを共有する必要があります。同じストレージレイヤーがない場合は、**nfs-provisioner** などのストレージプロバイダーを使用します。



注記

コンピュートプレーンとコントロールプレーン間のネットワークホップ数をできる限り制限する必要があります。

手順

1. OpenShift CLI (**oc**) にログインします。
2. 次のコマンドを実行すると、クラスターがアーキテクチャーペイロードを使用していることを確認できます。

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

検証

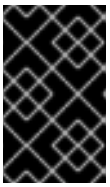
- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用しています。

```
{
 "release.openshift.io/architecture": "multi",
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```

  その後、クラスターへのマルチアーキテクチャーコンピュートノードの追加を開始できます。

- 次の出力が表示された場合、クラスターはマルチアーキテクチャーペイロードを使用していません。

```
{
 "url": "https://access.redhat.com/errata/<errata_version>"
}
```



重要

クラスターがマルチアーキテクチャーコンピューティングマシンをサポートするようにクラスターを移行するには、マルチアーキテクチャーコンピューティングマシンを 使用したクラスターへ の移行 の手順に従います。

4.9.2. ISO イメージを使用した RHCOS マシンの作成

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

前提条件

- クラスターのコンピュートマシンの Ignition 設定ファイルの URL を取得します。このファイルがインストール時に HTTP サーバーにアップロードされている必要があります。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 次のコマンドを実行して、クラスターから Ignition 設定ファイルを抽出します。

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --
to=- > worker.ign
```

2. クラスターからエクスポートした **worker.ign** Ignition 設定ファイルを HTTP サーバーにアップロードします。これらのファイルの URL をメモします。

3. Ignition ファイルが URL で利用可能であることを検証できます。次の例では、コンピュートノードの Ignition 設定ファイルを取得します。

```
$ curl -k http://<HTTP_server>/worker.ign
```

4. 次のコマンドを実行すると、新しいマシンを起動するための ISO イメージにアクセスできます。

```
RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get
configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.
<architecture>.artifacts.metal.formats.iso.disk.location')
```

5. ISO ファイルを使用して、追加のコンピュートマシンに RHCOS をインストールします。クラスターのインストール前にマシンを作成する際に使用したのと同じ方法を使用します。

   - ディスクに ISO イメージを書き込み、これを直接起動します。
   - LOM インターフェイスで ISO リダイレクトを使用します。

6. 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

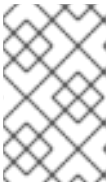
RHCOS インストールの起動プロセスを中断して、カーネル引数を追加できます。ただし、この ISO 手順では、カーネル引数を追加する代わりに、次の手順で概説するように **coreos-installer** コマンドを使用する必要があります。

7. **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> 1 2
```

1. **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。

2. **--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。



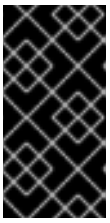
注記

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場合は、**coreos-installer** を実行する前に、内部認証局 (CA) をシステムのトラスト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
```

8. 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。

9. 継続してクラスター用の追加のコンピュートマシンを作成します。

4.9.3. PXE または iPXE ブートによる RHCOS マシンの作成

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

前提条件

- クラスターのコンピュートマシンの Ignition 設定ファイルの URL を取得します。このファイルがインストール時に HTTP サーバーにアップロードされている必要があります。
- クラスターのインストール時に HTTP サーバーにアップロードした RHCOS ISO イメージ、圧縮されたメタル BIOS、**kernel**、および **initramfs** ファイルの URL を取得します。
- インストール時に OpenShift Container Platform クラスターのマシンを作成するために使用した PXE ブートインフラストラクチャーにアクセスできる必要があります。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> (1)
    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 2
```

1. 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。
2. 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。

注記

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

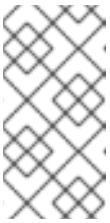
- iPXE (**x86_64** + **ppc64le**) の場合:

```
kernel http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> initrd=main
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign 1 2
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img 3
boot
```

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.live.rootfs_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.inst.ignition_url** パラメーター値は **rootfs** のライブファイルの場所です。
2. 複数の NIC を使用する場合、**ip** オプションに単一インターフェイスを指定します。たとえば、**eno1** という名前の NIC で DHCP を使用するには、**ip=eno1:dhcp** を設定します。
3. HTTP サーバーにアップロードした **initramfs** ファイルの場所を指定します。

注記

この設定では、グラフィカルコンソールを備えたマシンでのシリアルコンソールアクセスは有効になりません。別のコンソールを設定するには、**kernel** 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? と、「高度な RHCOS インストール設定」セクションの「PXE および ISO インストール用シリアルコンソールの有効化」を参照してください。



注記

ppc64le アーキテクチャーで CoreOS **kernel** をネットワークブートするには、**IMAGE_GZIP** オプションが有効になっているバージョンの iPXE ビルドを使用する必要があります。iPXE の **IMAGE_GZIP** オプション を参照してください。

- **ppc64le** 上の PXE (第 2 段階として UEFI と GRUB を使用) の場合:

```
menuentry 'Install CoreOS' {
    linux rhcos-<version>-live-kernel-<architecture>
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign 1 2
    initrd rhcos-<version>-live-initramfs.<architecture>.img 3
}
```

1. HTTP/TFTP 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、TFTP サーバー上の **kernel** ファイルの場所になります。**coreos.live.rootfs_url** パラメーター値は **rootfs** ファイルの場所であり、**coreos.inst.ignition_url** パラメーター値は HTTP サーバー上のブートストラップ Ignition 設定ファイルの場所になります。
2. 複数の NIC を使用する場合、**ip** オプションに単一インターフェイスを指定します。たとえば、**eno1** という名前の NIC で DHCP を使用するには、**ip=eno1:dhcp** を設定します。
3. TFTP 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。

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

4.9.4. 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

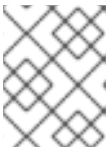
1. クラスターがマシンを認識していることを確認します。

```
$ oc get nodes
```

出力例

```
NAME      STATUS    ROLES   AGE  VERSION
master-0  Ready     master  63m  v1.29.4
master-1  Ready     master  63m  v1.29.4
master-2  Ready     master  64m  v1.29.4
```

出力には作成したすべてのマシンがリスト表示されます。



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。

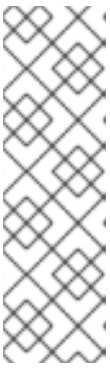
```
$ oc get csr
```

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。このリスト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 user-provisioned infrastructure 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 のリストからの 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 のリストからの 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 -o wide
```

出力例

```
NAME            STATUS   ROLES                  AGE   VERSION   INTERNAL-IP
EXTERNAL-IP   OS-IMAGE                                        KERNEL-VERSION
CONTAINER-RUNTIME
worker-0-ppc64le   Ready    worker                 42d   v1.29.5   192.168.200.21   <none>
Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.ppc64le   cri-o://1.29.5-3.rhaos4.15.gitb36169e.el9
worker-1-ppc64le   Ready    worker                 42d   v1.29.5   192.168.200.20   <none>
Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.ppc64le   cri-o://1.29.5-3.rhaos4.15.gitb36169e.el9
master-0-x86       Ready    control-plane,master   75d   v1.29.5   10.248.0.38      10.248.0.38
Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.x86_64    cri-o://1.29.5-3.rhaos4.15.gitb36169e.el9
master-1-x86       Ready    control-plane,master   75d   v1.29.5   10.248.0.39      10.248.0.39
Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.x86_64    cri-o://1.29.5-3.rhaos4.15.gitb36169e.el9
master-2-x86       Ready    control-plane,master   75d   v1.29.5   10.248.0.40      10.248.0.40
Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.x86_64    cri-o://1.29.5-3.rhaos4.15.gitb36169e.el9
worker-0-x86       Ready    worker                 75d   v1.29.5   10.248.0.43      10.248.0.43   Red
Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.x86_64    cri-o://1.29.5-3.rhaos4.15.gitb36169e.el9
worker-1-x86       Ready    worker                 75d   v1.29.5   10.248.0.44      10.248.0.44   Red
Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)   5.14.0-
284.34.1.el9_2.x86_64    cri-o://1.29.5-3.rhaos4.15.gitb36169e.el9
```

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。

4.10. マルチアーキテクチャーコンピュートマシンを含むクラスターの管理

4.10.1. マルチアーキテクチャーのコンピュートマシンを含むクラスターでワークロードをスケジュールする

さまざまなアーキテクチャーのコンピュートノードを含むクラスターにワークロードをデプロイするには、クラスターの注意と監視が必要です。クラスターのノードに Pod を正常に配置するには、さらにアクションが必要な場合があります。

Multiarch Tuning Operator を使用すると、マルチアーキテクチャーコンピュートマシンを含むクラスターで、アーキテクチャーを考慮したワークロードのスケジューリングを有効にできます。Multiarch Tuning Operator は、Pod が作成時にサポートできるアーキテクチャーに基づいて、Pod 仕様に追加のスケジューラー述語を実装します。詳細は、Multiarch Tuning Operator を使用してマルチアーキテクチャークラスター上のワークロードを管理する を参照してください。

ノードアフィニティー、スケジューリング、テイント、容認の詳細は、次のドキュメントを参照してください。

- ノードテイントを使用した Pod 配置の制御。
- ノードのアフィニティーを使用したノード上での Pod 配置の制御
- スケジューラーによる Pod 配置の制御

4.10.1.1. マルチアーキテクチャーノードのワークロードデプロイメントのサンプル

異なるアーキテクチャーのコンピュートノードを含むクラスター上でワークロードをスケジュールする前に、次の使用例を考慮してください。

ノードアフィニティーを使用してノード上のワークロードをスケジュールする

イメージによってサポートされるアーキテクチャーを持つ一連のノード上でのみワークロードをスケジュールできるようにすることができ、Pod のテンプレート仕様で **spec.affinity.nodeAffinity** フィールドを設定できます。

特定のアーキテクチャーに設定された nodeAffinity を使用したデプロイメントの例

```
apiVersion: apps/v1
kind: Deployment
metadata: # ...
spec:
  # ...
 template:
   # ...
  spec:
    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/arch
              operator: In
              values: 1
              - amd64
              - arm64
```

1 サポートされるアーキテクチャーを指定します。有効な値には、**amd64**、**arm64**、または両方の値が含まれます。

特定のアーキテクチャー向けにすべてのノードをテイントする

ノードをテイントして、そのノード上でそのアーキテクチャーと互換性のないワークロードがスケジュールされるのを回避できます。クラスターが **MachineSet** オブジェクトを使用している場合は、サポートされていないアーキテクチャーのノードでワークロードがスケジュールされるのを回避するために、パラメーターを **.spec.template.spec.taints** フィールドに追加できます。

- ノードをテイントする前に、**MachineSet** オブジェクトをスケールダウンするか、使用可能なマシンを削除する必要があります。次のコマンドのいずれかを使用して、マシンセットをスケールダウンできます。

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

  または、以下を実行します。

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

  マシンセットのスケーリングの詳細は、「コンピュートマシンセットの変更」を参照してください。

テイントセットを含む MachineSet の例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata: # ...
spec:
  # ...
  template:
    # ...
    spec:
      # ...
      taints:
      - effect: NoSchedule
        key: multi-arch.openshift.io/arch
        value: arm64
```

次のコマンドを実行して、特定のノードにテイントを設定することもできます。

```
$ oc adm taint nodes <node-name> multi-arch.openshift.io/arch=arm64:NoSchedule
```

デフォルト許容範囲の作成

次のコマンドを実行して、namespace にアノテーションを付けて、すべてのワークロードが同じデフォルトの許容範囲を取得できるようにすることができます。

```
$ oc annotate namespace my-namespace \
  'scheduler.alpha.kubernetes.io/defaultTolerations'='[{"operator": "Exists", "effect": "NoSchedule",
"key": "multi-arch.openshift.io/arch"}]'
```

ワークロードにおけるアーキテクチャーのテイントを許容する

テイントが定義されたノードでは、そのノード上でワークロードはスケジュールされません。ただし、Pod の仕様で容認を設定することで、スケジュールを許可できます。

許容を備えた導入例

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata: # ...
spec:
  # ...
  template:
    # ...
    spec:
      tolerations:
      - key: "multi-arch.openshift.io/arch"
        value: "arm64"
        operator: "Equal"
        effect: "NoSchedule"
```

このデプロイメント例は、**multi-arch.openshift.io/arch=arm64** テイントが指定されたノードでも許可できます。

テイントおよび許容でのノードアフィニティーの使用

スケジューラーが Pod をスケジュールするためにノードのセットを計算する場合は、ノードアフィニティーによってセットが制限される一方で、許容によってセットが拡大する可能性があります。特定のアーキテクチャーのノードにテイントを設定する場合、Pod のスケジュールには次の例の許容が必要です。

ノードアフィニティーと許容セットを使用したデプロイメントの例。

```
apiVersion: apps/v1
kind: Deployment
metadata: # ...
spec:
  # ...
  template:
    # ...
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/arch
                operator: In
                values:
                - amd64
                - arm64
      tolerations:
      - key: "multi-arch.openshift.io/arch"
        value: "arm64"
        operator: "Equal"
        effect: "NoSchedule"
```

関連情報

- コンピュートマシンセットの変更

4.10.2. Red Hat Enterprise Linux CoreOS (RHCOS) カーネルでの 64k ページの有効化

クラスター内の 64 ビット ARM コンピュートマシン上の Red Hat Enterprise Linux CoreOS (RHCOS) カーネルで 64k メモリーページを有効にすることができます。64k ページサイズのカーネル仕様は、大規模な GPU または高メモリーのワークロードに使用できます。これは、マシン設定プールを使用してカーネルを更新する Machine Config Operator (MCO) を使用して行われます。64k ページサイズを有効にするには、ARM64 専用のマシン設定プールをカーネルで有効にする必要があります。

重要

64k ページの使用は、64 ビット ARM マシンにインストールされた 64 ビット ARM アーキテクチャーのコンピュートノードまたはクラスターに限定されます。64 ビット x86 マシンを使用してマシン設定プールに 64k ページのカーネルを設定すると、マシン設定プールと MCO がデグレード状態になります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- サポート対象のいずれかのプラットフォームで、異なるアーキテクチャーのコンピュートノードを含むクラスターを作成している。

手順

1. 64k ページサイズのカーネルを実行するノードにラベルを付けます。

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

コマンドの例

```
$ oc label node worker-arm64-01 node-role.kubernetes.io/worker-64k-pages=
```

2. ARM64 アーキテクチャーを使用するワーカーロールと **worker-64k-pages** ロールを含むマシン設定プールを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker-64k-pages
spec:
  machineConfigSelector:
    matchExpressions:
      - key: machineconfiguration.openshift.io/role
        operator: In
        values:
        - worker
        - worker-64k-pages
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker-64k-pages: ""
      kubernetes.io/arch: arm64
```

3. コンピュートノード上にマシン設定を作成し、**64k-pages** パラメーターを使用して **64k-pages** を有効にします。

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

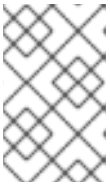
-

MachineConfig の例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "worker-64k-pages" (1)
  name: 99-worker-64kpages
spec:
  kernelType: 64k-pages (2)
```

(1) カスタムマシン設定プールの **machineconfiguration.openshift.io/role** ラベルの値を指定します。MachineConfig の例では、**worker-64k-pages** ラベルを使用して、**worker-64k-pages** プールで 64k ページを有効にしています。

(2) 必要なカーネルタイプを指定します。有効な値は **64k-pages** と **default** です。



注記

64k-pages タイプは、64 ビット ARM アーキテクチャーベースのコンピュートノードでのみサポートされます。**realtime** タイプは、64 ビット x86 アーキテクチャーベースのコンピュートノードでのみサポートされます。

検証

- 新しい **worker-64k-pages** マシン設定プールを表示するには、次のコマンドを実行します。

```
$ oc get mcp
```

出力例

```
NAME     CONFIG                                             UPDATED   UPDATING
DEGRADED   MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-9d55ac9a91127c36314e1efe7d77fbf8                 True      False
False      3            3                   3                   0                   361d
worker   rendered-worker-e7b61751c4a5b7ff995d64b967c421ff                 True      False
False      7            7                   7                   0                   361d
worker-64k-pages  rendered-worker-64k-pages-e7b61751c4a5b7ff995d64b967c421ff   True
False      False      2            2                   2                   0                   35m
```

4.10.3. マルチアーキテクチャーコンピュートマシンのイメージストリームにマニフェストリストをインポートする

マルチアーキテクチャーコンピュートマシンを持つ OpenShift Container Platform 4.16 クラスターでは、クラスター内のイメージストリームはマニフェストリストを自動的にインポートしません。マニフェストリストをインポートするには、デフォルトの **importMode** オプションを **PreserveOriginal** オプションに手動で変更する必要があります。

前提条件

- OpenShift Container Platform CLI (**oc**) をインストールしている。

手順

- 次のコマンド例は、**ImageStream** cli-artifacts にパッチを適用して、**cli-artifacts:latest** イメージストリームタグがマニフェストリストとしてインポートされるようにする方法を示しています。

```
$ oc patch is/cli-artifacts -n openshift -p '{"spec":{"tags":[{"name":"latest","importPolicy":
{"importMode":"PreserveOriginal"}}]}}'
```

検証

- イメージストリームタグを調べて、マニフェストリストが正しくインポートされたことを確認できます。次のコマンドは、特定のタグの個々のアーキテクチャーマニフェストを一覧表示します。

```
$ oc get istag cli-artifacts:latest -n openshift -oyaml
```

dockerImageManifests オブジェクトが存在する場合、マニフェストリストのインポートは成功しています。

dockerImageManifests オブジェクトの出力例

```
dockerImageManifests:
  - architecture: amd64
    digest:
sha256:16d4c96c52923a9968fbfa69425ec703aff711f1db822e4e9788bf5d2bee5d77
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
  - architecture: arm64
    digest:
sha256:6ec8ad0d897bcdf727531f7d0b716931728999492709d19d8b09f0d90d57f626
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
  - architecture: ppc64le
    digest:
sha256:65949e3a80349cdc42acd8c5b34cde6ebc3241eae8daaeea458498fedb359a6a
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
  - architecture: s390x
    digest:
sha256:75f4fa21224b5d5d511bea8f92dfa8e1c00231e5c81ab95e83c3013d245d1719
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
```

4.11. MULTIARCH TUNING OPERATOR を使用してマルチアーキテクチャークラスター上のワークロードを管理する

重要

Multiarch Tuning Operator はテクノロジープレビューのみの機能です。テクノロジープレビュー機能は、Red Hat 製品サポート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 では、実稼働環境での使用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。

Multiarch Tuning Operator は、マルチアーキテクチャークラスター、およびマルチアーキテクチャーコンピュート設定に移行中のシングルアーキテクチャークラスターにおける運用エクスペリエンスを強化します。

この Operator は、アーキテクチャーを考慮したワークロードのスケジューリングをサポートするために、**clusterpodplacementconfigs** カスタムリソース (CR) を実装します。

アーキテクチャーを考慮したワークロードのスケジューリングを有効にするには、**ClusterPodPlacementConfig** オブジェクトを作成する必要があります。**ClusterPodPlacementConfig** オブジェクトを作成すると、この Operator がオペランドをデプロイします。

Pod が作成されると、オペランドは次のアクションを実行します。

1. Pod のスケジュールを防止するスケジューリングゲート **multiarch.openshift.io/scheduling-gate** を追加します。
2. **kubernetes.io/arch** ラベルでサポートされているアーキテクチャー値を含むスケジューリング述語を計算します。
3. スケジューリング述語を Pod 仕様の **nodeAffinity** 要件として統合します。
4. Pod からスケジューリングゲートを削除します。

オペランドがスケジューリングゲートを削除すると、Pod はスケジューリングサイクルに入ります。次に、サポートされているアーキテクチャーに基づいて、ノード上でワークロードがスケジュールされます。

重要

テクノロジープレビューバージョンの Multiarch Tuning Operator は、ネットワークが制限された環境のクラスターをサポートしていません。

4.11.1. CLI を使用した Multiarch Tuning Operator のインストール

OpenShift CLI (**oc**) を使用して、Multiarch Tuning Operator をインストールできます。

前提条件

- **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 **oc** にログインしている。

手順

1. 次のコマンドを実行して、**openshift-multiarch-tuning-Operator** という名前の新しいプロジェクトを作成します。

```
$ oc create ns openshift-multiarch-tuning-operator
```

2. **OperatorGroup** オブジェクトを作成します。
   a. **OperatorGroup** オブジェクトを作成するための設定を含む YAML ファイルを作成します。

   OperatorGroup オブジェクトを作成するための YAML 設定の例:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-multiarch-tuning-operator
  namespace: openshift-multiarch-tuning-operator
spec: {}
```

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

```
$ oc create -f <file_name> 1
```

   1 **<file_name>** は、**OperatorGroup** オブジェクトの設定を含む YAML ファイルの名前に置き換えます。

3. **Subscription** オブジェクトを作成します。
   a. **Subscription** オブジェクトを作成するための設定を含む YAML ファイルを作成します。

   Subscription オブジェクトを作成するための YAML 設定の例:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-multiarch-tuning-operator
  namespace: openshift-multiarch-tuning-operator
spec:
  channel: tech-preview
  name: multiarch-tuning-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Automatic
  startingCSV: multiarch-tuning-operator.v0.9.0
```

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

```
$ oc create -f <file_name> 1
```

   1 **<file_name>** は、**Subscription** オブジェクトの設定を含む YAML ファイルの名前に置き換えます。

注記

Subscription オブジェクトと **OperatorGroup** オブジェクトの設定の詳細は、「CLI を使用した OperatorHub からのインストール」を参照してください。

検証

1. Multiarch Tuning Operator がインストールされていることを確認するには、次のコマンドを実行します。

```
$ oc get csv -n openshift-multiarch-tuning-operator
```

出力例

```
NAME                                DISPLAY                  VERSION   REPLACES   PHASE
multiarch-tuning-operator.v0.9.0    Multiarch Tuning Operator   0.9.0                Succeeded
```

Operator が **Succeeded** フェーズにある場合、インストールは成功しています。

2. オプション: **OperatorGroup** オブジェクトが作成されたことを確認するには、次のコマンドを実行します。

```
$ oc get operatorgroup -n openshift-multiarch-tuning-operator
```

出力例

```
NAME                                AGE
openshift-multiarch-tuning-operator-q8zbb   133m
```

3. オプション: **Subscription** オブジェクトが作成されたことを確認するには、次のコマンドを実行します。

```
$ oc get subscription -n openshift-multiarch-tuning-operator
```

出力例

```
NAME                        PACKAGE                     SOURCE                          CHANNEL
multiarch-tuning-operator   multiarch-tuning-operator   multiarch-tuning-operator-catalog   tech-preview
```

関連情報

- CLI を使用した OperatorHub からのインストール

4.11.2. Web コンソールを使用した Multiarch Tuning Operator のインストール

OpenShift Container Platform Web コンソールを使用して、Multiarch Tuning Operator をインストールできます。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。

- OpenShift Container Platform Web コンソールにアクセスできる。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operators → OperatorHub** に移動します。
3. 検索フィールドに **Multiarch Tuning Operator** と入力します。
4. **Multiarch Tuning Operator** をクリックします。
5. **Version** リストから **Multiarch Tuning Operator** のバージョンを選択します。
6. **Install** をクリックします。
7. **Operator Installation** ページで次のオプションを設定します。
   a. **Update Channel** を **tech-preview** に設定します。
   b. **Installation Mode** を **All namespaces on the cluster**に設定します。
   c. **Installed Namespace** を、**Operator recommended Namespace** か **Select a Namespace** に設定します。
   推奨される Operator namespace は **openshift-multiarch-tuning-operator** です。**openshift-multiarch-tuning- operator** namespace が存在しない場合は、Operator のインストール中に作成されます。

   Select a namespace を選択した場合は、**Select Project** リストから Operator の namespace を選択する必要があります。
   d. **Update approval** で **Automatic** または **Manual** を選択します。
   Automatic 更新を選択すると、Operator Lifecycle Manager (OLM) が管理者の介入なしで Multiarch Tuning Operator の実行中のインスタンスを自動的に更新します。

   Manual 更新を選択すると、OLM は更新要求を作成します。クラスター管理者は、Multiarch Tuning Operator を新しいバージョンに更新するために、更新要求を手動で承認する必要があります。
8. オプション: **Enable Operator recommended cluster monitoring on this Namespace**チェックボックスを選択します。
9. **Install** をクリックします。

検証

1. **Operators → Installed Operators** に移動します。
2. **Multiarch Tuning Operator** が、**openshift-multiarch-tuning-operator** namespace の **Status** フィールドに **Succeeded** としてリストされていることを確認します。

4.11.3. ClusterPodPlacementConfig オブジェクトの作成

Multiarch Tuning Operator をインストールしたら、**ClusterPodPlacementConfig** オブジェクトを作成する必要があります。このオブジェクトを作成すると、Multiarch Tuning Operator が、アーキテクチャーを考慮したワークロードスケジューリングを可能にするオペランドをデプロイします。

注記

ClusterPodPlacementConfig オブジェクトのインスタンスは 1 つだけ作成できます。

ClusterPodPlacementConfig オブジェクトの設定例

```
apiVersion: multiarch.openshift.io/v1beta1
kind: ClusterPodPlacementConfig
metadata:
  name: cluster 1
spec:
  logVerbosityLevel: Normal 2
  namespaceSelector: 3
    matchExpressions:
      - key: multiarch.openshift.io/exclude-pod-placement
        operator: DoesNotExist
```

1 このフィールドの値を **cluster** に設定する必要があります。

2 オプション: フィールドの値を **Normal**、**Debug**、**Trace**、または **TraceAll** に設定できます。デフォルトでは、値は **Normal** に設定されています。

3 オプション: **namespaceSelector** を設定すると、Multiarch Tuning Operator の Pod 配置オペランドによって Pod の **nodeAffinity** を処理する必要がある namespace を選択できます。デフォルトではすべての namespace が考慮されます。

この例では、**operator** フィールドの値が **DoesNotExist** に設定されています。したがって、**key** フィールドの値 (**multiarch.openshift.io/exclude-pod-placement**) が namespace のラベルとして設定されている場合、オペランドはその namespace 内の Pod の **nodeAffinity** を処理しません。代わりに、オペランドはこのラベルを含まない namespace 内の Pod の **nodeAffinity** を処理します。

オペランドで特定の namespace 内の Pod の **nodeAffinity** のみを処理する場合は、次のように **namespaceSelector** を設定できます。

```
namespaceSelector:
  matchExpressions:
    - key: multiarch.openshift.io/include-pod-placement
      operator: Exists
```

この例では、**operator** フィールドの値が **Exists** に設定されています。したがって、オペランドは、**multiarch.openshift.io/include-pod-placement** ラベルを含む namespace 内の Pod の **nodeAffinity** のみを処理します。

重要

openshift-、**kube-**、**hypershift-** で始まる namespace は除外されます。

4.11.3.1. CLI を使用した ClusterPodPlacementConfig オブジェクトの作成

アーキテクチャーを考慮したワークロードのスケジューリングを可能にする Pod 配置オペランドをデプロイするには、OpenShift CLI (**oc**) を使用して **ClusterPodPlacementConfig** オブジェクトを作成します。

前提条件

- **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 **oc** にログインしている。
- Multiarch Tuning Operator がインストールされている。

手順

1. **ClusterPodPlacementConfig** オブジェクトの YAML ファイルを作成します。

   ClusterPodPlacementConfig オブジェクトの設定例

```
apiVersion: multiarch.openshift.io/v1beta1
kind: ClusterPodPlacementConfig
metadata:
  name: cluster
spec:
  logVerbosityLevel: Normal
  namespaceSelector:
    matchExpressions:
      - key: multiarch.openshift.io/exclude-pod-placement
        operator: DoesNotExist
```

2. 次のコマンドを実行して **ClusterPodPlacementConfig** オブジェクトを作成します。

```
$ oc create -f <file_name> 1
```

   1 **<file_name>** は、**ClusterPodPlacementConfig** オブジェクトの YAML ファイルの名前に置き換えます。

検証

- **ClusterPodPlacementConfig** オブジェクトが作成されたことを確認するには、次のコマンドを実行します。

```
$ oc get clusterpodplacementconfig
```

   出力例

```
NAME      AGE
cluster   29s
```

4.11.3.2. Web コンソールを使用した ClusterPodPlacementConfig オブジェクトの作成

アーキテクチャーを考慮したワークロードのスケジューリングを可能にする Pod 配置オペランドをデプロイするには、OpenShift Container Platform Web コンソールを使用して **ClusterPodPlacementConfig** オブジェクトを作成します。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。
- Multiarch Tuning Operator がインストールされている。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operators → Installed Operators** に移動します。
3. **Installed Operators** ページで、**Multiarch Tuning Operator** をクリックします。
4. **Cluster Pod Placement Config** タブをクリックします。
5. **Form view** または **YAML view** のいずれかを選択します。
6. **ClusterPodPlacementConfig** オブジェクトのパラメーターを設定します。
7. **Create** をクリックします。
8. オプション: **ClusterPodPlacementConfig** オブジェクトを編集する場合は、次の操作を実行します。
    a. **Cluster Pod Placement Config** タブをクリックします。
    b. オプションメニューから **Edit ClusterPodPlacementConfig** を選択します。
    c. **YAML** をクリックし、**ClusterPodPlacementConfig** オブジェクトのパラメーターを編集します。
    d. **Save** をクリックします。

検証

- **Cluster Pod Placement Config** ページで、**ClusterPodPlacementConfig** オブジェクトが **Ready** 状態であることを確認します。

4.11.4. CLI を使用した ClusterPodPlacementConfig オブジェクトの削除

ClusterPodPlacementConfig オブジェクトのインスタンスは 1 つだけ作成できます。このオブジェクトを再作成する場合は、まず既存のインスタンスを削除する必要があります。

OpenShift CLI (**oc**) を使用してこのオブジェクトを削除できます。

前提条件

- **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 **oc** にログインしている。

手順

1. OpenShift CLI (**oc**) にログインします。

2. 次のコマンドを実行して **ClusterPodPlacementConfig** オブジェクトを削除します。

```
$ oc delete clusterpodplacementconfig cluster
```

検証

- **ClusterPodPlacementConfig** オブジェクトが削除されたことを確認するには、次のコマンドを実行します。

```
$ oc get clusterpodplacementconfig
```

出力例

```
No resources found
```

次のステップ

- **ClusterPodPlacementConfig** オブジェクトを削除したら、**SchedulingGated** が原因で Pod が **Pending** フェーズになっていないことを確認します。次のコマンドを実行すると、ゲートが設定されたすべての Pod からスケジューリングゲートを削除できます。

```
$ oc get pods -A -l multiarch.openshift.io/scheduling-gate=gated -o json  | jq
'del(.items[].spec.schedulingGates[] | select(.name=="multiarch.openshift.io/scheduling-
gate"))' | oc apply -f -
```

4.11.5. Web コンソールを使用した ClusterPodPlacementConfig オブジェクトの削除

ClusterPodPlacementConfig オブジェクトのインスタンスは 1 つだけ作成できます。このオブジェクトを再作成する場合は、まず既存のインスタンスを削除する必要があります。

OpenShift Container Platform Web コンソールを使用してこのオブジェクトを削除できます。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。
- **ClusterPodPlacementConfig** オブジェクトを作成した。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operators → Installed Operators** に移動します。
3. **Installed Operators** ページで、**Multiarch Tuning Operator** をクリックします。
4. **Cluster Pod Placement Config** タブをクリックします。
5. オプションメニューから **Delete ClusterPodPlacementConfig** を選択します。
6. **Delete** をクリックします。

検証

- Cluster Pod Placement Config ページで、**ClusterPodPlacementConfig** オブジェクトが削除されていることを確認します。

次のステップ

- **ClusterPodPlacementConfig** オブジェクトを削除したら、**SchedulingGated** が原因で Pod が **Pending** フェーズになっていないことを確認します。OpenShift CLI (**oc**) で次のコマンドを実行すると、ゲートが設定されたすべての Pod からスケジューリングゲートを削除できます。

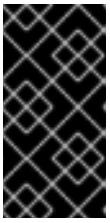
```
$ oc get pods -A -l multiarch.openshift.io/scheduling-gate=gated -o json  | jq
'del(.items[].spec.schedulingGates[] | select(.name=="multiarch.openshift.io/scheduling-
gate"))' | oc apply -f -
```

4.11.6. CLI を使用した Multiarch Tuning Operator のアンインストール

OpenShift CLI (**oc**) を使用して、Multiarch Tuning Operator をアンインストールできます。

前提条件

- **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 **oc** にログインしている。
- **ClusterPodPlacementConfig** オブジェクトを削除した。



重要

Multiarch Tuning Operator をアンインストールする前に、**ClusterPodPlacementConfig** オブジェクトを削除する必要があります。**ClusterPodPlacementConfig** オブジェクトを削除せずに Operator をアンインストールすると、予期しない動作が発生する可能性があります。

手順

1. 次のコマンドを実行して、Multiarch Tuning Operator の **currentCSV** 値を取得します。

```
$ oc get subscription.operators.coreos.com multiarch-tuning-operator -n <namespace> -o
yaml | grep currentCSV (1)
```

(1) **<namespace>** は、Multiarch Tuning Operator をアンインストールする namespace の名前に置き換えます。

出力例

```
currentCSV: multiarch-tuning-operator.v0.9.0
```

2. 次のコマンドを実行して、**Subscription** オブジェクトを削除します。

```
$ oc delete subscription.operators.coreos.com openshift-multiarch-tuning-operator -n
<namespace> (1)
```

1 **<namespace>** は、Multiarch Tuning Operator をアンインストールする namespace の名前に置き換えます。

出力例

```
subscription.operators.coreos.com "openshift-multiarch-tuning-operator" deleted
```

3. 次のコマンドを実行して、**currentCSV** 値を使用してターゲット namespace 内の Multiarch Tuning Operator の CSV を削除します。

```
$ oc delete clusterserviceversion <currentCSV_value> -n <namespace> 1
```

1 **<currentCSV>** は、Multiarch Tuning Operator の **currentCSV** 値に置き換えます。たとえば、**multiarch-tuning-operator.v0.9.0** です。**<namespace>** は、Multiarch Tuning Operator をアンインストールする namespace の名前に置き換えます。

出力例

```
clusterserviceversion.operators.coreos.com "multiarch-tuning-operator.v0.9.0" deleted
```

検証

- Multiarch Tuning Operator がアンインストールされたことを確認するには、次のコマンドを実行します。

```
$ oc get csv -n <namespace> 1
```

1 **<namespace>** は、Multiarch Tuning Operator をアンインストールした namespace の名前に置き換えます。

出力例

```
No resources found in openshift-multiarch-tuning-operator namespace.
```

4.11.7. Web コンソールを使用した Multiarch Tuning Operator のアンインストール

OpenShift Container Platform Web コンソールを使用して、Multiarch Tuning Operator をアンインストールできます。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできます。
- **ClusterPodPlacementConfig** オブジェクトを削除した。

重要

Multiarch Tuning Operator をアンインストールする前に、**ClusterPodPlacementConfig** オブジェクトを削除する必要があります。**ClusterPodPlacementConfig** オブジェクトを削除せずに Operator をアンインストールすると、予期しない動作が発生する可能性があります。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operators → OperatorHub** に移動します。
3. 検索フィールドに **Multiarch Tuning Operator** と入力します。
4. **Multiarch Tuning Operator** をクリックします。
5. **Details** タブをクリックします。
6. **Actions** メニューから、**Uninstall Operator** を選択します。
7. プロンプトが表示されたら、**Uninstall** をクリックします。

検証

1. **Operators → Installed Operators** に移動します。
2. **Installed Operator** ページで、**Multiarch Tuning Operator** がリストされていないことを確認します。

第5章 VSPHERE クラスターでの暗号化の有効化

vSphere に OpenShift Container Platform 4.16 をインストールした後、ノードを 1 つずつドレインしてシャットダウンすることにより、仮想マシンを暗号化できます。各仮想マシンがシャットダウンしている間、vCenter Web インターフェイスで暗号化を有効にすることができます。

5.1. 仮想マシンの暗号化

次のプロセスで仮想マシンを暗号化できます。vCenter インターフェイスを使用して、仮想マシンをドレインし、電源を切り、暗号化することができます。最後に、暗号化されたストレージを使用するストレージクラスを作成できます。

前提条件

- vSphere で標準キープロバイダーを設定しました。詳細については、vCenter Server への KMS の追加 を参照してください。

重要

vCenter のネイティブキープロバイダーはサポートされていません。詳細については、vSphere Native Key Provider の概要 を参照してください。

- クラスターをホスティングしているすべての ESXi ホストでホスト暗号化モードを有効にしました。詳細については、ホスト暗号化モードの有効化 を参照してください。
- すべての暗号化権限が有効になっている vSphere アカウントがあります。詳細については、暗号化操作の権限 を参照してください。

手順

1. ノードの 1 つをドレインして閉鎖します。ノード管理の詳細な手順については、「ノードの操作」を参照してください。
2. vCenter インターフェイスでそのノードに関連付けられている仮想マシンをシャットダウンします。
3. vCenter インターフェイスで仮想マシンを右クリックし、**VM Policies → Edit VM Storage Policies** を選択します。
4. 暗号化されたストレージポリシーを選択し、**OK** を選択します。
5. vCenter インターフェイスで暗号化された仮想マシンを起動します。
6. 暗号化するすべてのノードに対して、手順 1~5 を繰り返します。
7. 暗号化されたストレージポリシーを使用するストレージクラスを設定します。暗号化されたストレージクラスの設定の詳細については、"VMware vSphere CSI Driver Operator" を参照してください。

5.2. 関連情報

- ノードの使用
- vSphere 暗号化

- 仮想マシンを暗号化するための要件

第6章 インストール後の VSPHERE 接続設定

インストール方法によっては、プラットフォーム統合機能を有効にして OpenShift Container Platform クラスターを vSphere にインストールした後に、vSphere 接続設定を手動で更新する必要があります。

アシステッドインストーラーを使用したインストールの場合は、接続設定を更新する必要があります。これは、インストール時にアシステッドインストーラーが、デフォルトの接続設定をプレースホルダーとして **vSphere connection configuration** ウィザードに追加するためです。

インストーラーまたはユーザーによってプロビジョニングされるインフラストラクチャーをインストールする場合は、インストール時に有効な接続設定を入力する必要があります。**vSphere connection configuration** ウィザードは任意のタイミングで使用して接続設定を検証または変更できますが、これはインストールの完了に必須の操作ではありません。

6.1. VSPHERE 接続設定

必要に応じて、以下の vSphere 設定を変更します。

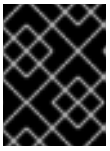
- vCenter アドレス
- vCenter クラスター
- vCenter ユーザー名
- vCenter パスワード
- vCenter アドレス
- vSphere データセンター
- vSphere データストア
- 仮想マシンフォルダー

前提条件

- アシステッドインストーラーによってクラスターが正常にインストールされている。
- クラスターが **https://console.redhat.com** に接続されている。

手順

1. Administrator パースペクティブで、**Home → Overview** に移動します。
2. **Status** で **vSphere connection** をクリックし、**vSphere connection configuration** ウィザードを開きます。
3. **vCenter** フィールドに、vSphere vCenter サーバーのネットワークアドレスを入力します。ドメイン名または IP アドレスのいずれかを入力できます。これは vSphere Web クライアント URL に表示されます (例: **https://[your_vCenter_address]/ui**)。
4. **vCenter クラスター** フィールドには、OpenShift Container Platform がインストールされている vSphere vCenter クラスターの名前を入力します。



重要

この手順は、OpenShift Container Platform 4.13 以降をインストールしている場合は必須となります。

5. **Username** フィールドに、vSphere vCenter のユーザー名を入力します。
6. **Password** フィールドに、vSphere vCenter のパスワードを入力します。

警告

システムは、クラスターの **kube-system** namespace の **vsphere-creds** シークレットにユーザー名とパスワードを保存します。vCenter のユーザー名またはパスワードが間違っていると、クラスターノードをスケジュールできなくなります。

7. **Datacenter** フィールドに、クラスターのホストに使用する仮想マシンが含まれる vSphere データセンターの名前を入力します (例: **SDDC-Datacenter**)。
8. **Default data store** フィールドに、永続データボリュームを保存する vSphere データストアのパスおよび名前を入力します (例: **/SDDC-Datacenter/datastore/datastorename**)。

警告

設定の保存後に vSphere データセンターまたはデフォルトのデータストアを更新すると、アクティブな vSphere **PersistentVolumes** がデタッチされます。

9. **Virtual Machine Folder** フィールドに、クラスターの仮想マシンが含まれるデータセンターフォルダーを入力します (例: **/SDDC-Datacenter/vm/ci-ln-hjg4vg2-c61657-t2gzr**)。正常に OpenShift Container Platform をインストールするには、クラスターを構成するすべての仮想マシンを単一のデータセンターフォルダーに配置する必要があります。
10. **Save Configuration** をクリックします。これにより、**openshift-config** namespace の **cloud-provider-config** ConfigMap リソースが更新され、設定プロセスが開始されます。
11. **vSphere connection configuration** ウィザードを再度開き、**Monitored operators** パネルを展開します。Operator のステータスが **Progressing** または **Healthy** であることを確認します。

6.2. 設定の確認

接続設定プロセスは、Operator ステータスとコントロールプレーンノードを更新します。完了するまでに約 1 時間かかります。設定プロセスの中でノードが再起動します。これまでは、バインドされた **PersistentVolumeClaims** オブジェクトの接続が切断される可能性がありました。

前提条件

- **vSphere connection configuration** 設定ウィザードで設定を保存している。

手順

1. 設定プロセスが正常に完了したことを確認します。

   a. OpenShift Container Platform の Administrator パースペクティブで、**Home → Overview** に移動します。

   b. **Status** で **Operators** をクリックします。すべての Operator ステータスが **Progressing** から **All succeeded** に変わるまで待機します。**Failed** ステータスは、設定が失敗したことを示します。

   c. **Status** で **Control Plane** をクリックします。すべての Control Pane コンポーネントの応答レートが 100% に戻るまで待機します。**Failed** コントロールプレーンコンポーネントは、設定が失敗したことを示します。

   失敗は、少なくとも 1 つの接続設定が間違っていることを示します。**vSphere connection configuration** ウィザードで設定を変更し、その設定を再度保存します。

2. 以下の手順を実行して、**PersistentVolumeClaims** オブジェクトをバインドできることを確認します。

   a. 以下の YAML を使用して **StorageClass** オブジェクトを作成します。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: vsphere-sc
provisioner: kubernetes.io/vsphere-volume
parameters:
 datastore: YOURVCENTERDATASTORE
 diskformat: thin
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

   b. 以下の YAML を使用して **PersistentVolumeClaims** オブジェクトを作成します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: test-pvc
 namespace: openshift-config
 annotations:
   volume.beta.kubernetes.io/storage-provisioner: kubernetes.io/vsphere-volume
 finalizers:
   - kubernetes.io/pvc-protection
spec:
 accessModes:
   - ReadWriteOnce
 resources:
   requests:
```

```
    storage: 10Gi
  storageClassName: vsphere-sc
  volumeMode: Filesystem
```

PersistentVolumeClaims オブジェクトを作成できない場合、OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで **Storage** → **PersistentVolumeClaims** に移動してトラブルシューティングできます。

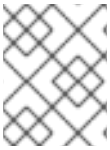
ストレージオブジェクトの作成手順については、動的プロビジョニング を参照してください。

第7章 インストール後のクラスタータスク

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

7.1. 利用可能なクラスターのカスタマイズ

OpenShift Container Platform クラスターのデプロイ後は、大半のクラスター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

クラスターを 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ープロバイダーなどのクラスター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**)。

7.1.1. クラスター設定リソース

すべてのクラスター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	クラスターの アイデンティティープロバイダー および認証設定を制御します。
build.config.openshift.io	クラスター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 の処理方法を設定します(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ー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されるプロキシーを定義します。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レクターなどの スケジューラー の動作を設定します。

7.1.2. 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、プロキシー 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。

7.1.3. 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
ingresscontroller.operator.openshift.io	**default**	**openshift-ingress-operator**	ドメイン、レプリカの数、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。

7.1.4. 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
clusterversion.config.openshift.io	**version**	OpenShift Container Platform 4.16 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、クラスターの更新 プロセスを実行します。
dns.config.openshift.io	**cluster**	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。
infrastructure.config.openshift.io	**cluster**	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。
network.config.openshift.io	**cluster**	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。

7.2. 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> 1
```

1 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>" \ 1
--auth-basic="<username>:<password>" \ 2
--to=<pull_secret_location> 3
```

1 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ーを含めることができます (例: **--registry="<registry/my-namespace/my-repository>"**)。

2 新しいレジストリーの認証情報を指定します。

3 プルシークレットファイルへのパスを指定します。

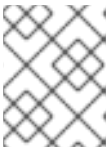
または、プルシークレットファイルを手動で更新することもできます。

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> 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。

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。

7.3. ワーカーノードの追加

OpenShift Container Platform クラスターをデプロイしたら、ワーカーノードを追加してクラスターリソースをスケーリングできます。インストール方法とクラスターの環境に応じて、ワーカーノードを追加するさまざまな方法があります。

7.3.1. installer-provisioned infrastructure へのワーカーノードの追加

installer-provisioned infrastructure クラスターの場合、**MachineSet** オブジェクトを手動または自動でスケーリングして、利用可能なベアメタルホストの数に一致させることができます。

ベアメタルホストを追加するには、すべてのネットワーク前提条件を設定し、関連する **baremetalhost** オブジェクトを設定してから、クラスターにワーカーノードをプロビジョニングする必要があります。手動で、または Web コンソールを使用して、ベアメタルホストを追加できます。

- Web コンソールを使用したワーカーノードの追加

- Web コンソールで YAML を使用したワーカーノードの追加
- installer-provisioned infrastructure クラスターへのワーカーノードの手動での追加

7.3.2. user-provisioned infrastructure クラスターへのワーカーノードの追加

user-provisioned infrastructure クラスターの場合、RHEL または RHCOS ISO イメージを使用し、クラスター Ignition 設定ファイルを使用してこれをクラスターに接続することで、ワーカーノードを追加できます。RHEL ワーカーノードの場合、次の例では、Ansible Playbook を使用してクラスターにワーカーノードを追加します。RHCOS ワーカーノードの場合、次の例では、ISO イメージとネットワークブートを使用してワーカーノードをクラスターに追加します。

- user-provisioned infrastructure クラスターへの RHCOS ワーカーノードの追加
- user-provisioned infrastructure クラスターへの RHEL ワーカーノードの追加

7.3.3. Assisted Installer によって管理されるクラスターへのワーカーノードの追加

Assisted Installer によって管理されるクラスターの場合、Red Hat OpenShift Cluster Manager コンソール、Assisted Installer REST API を使用してワーカーノードを追加するか、ISO イメージとクラスター Ignition 設定ファイルを使用してワーカーノードを手動で追加することができます。

- OpenShift Cluster Manager を使用したワーカーノードの追加
- Assisted Installer REST API を使用したワーカーノードの追加
- 手動でのワーカーノードの SNO クラスターへの追加

7.3.4. Kubernetes のマルチクラスターエンジンによって管理されるクラスターへのワーカーノードの追加

Kubernetes のマルチクラスターエンジンによって管理されるクラスターの場合、専用のマルチクラスターエンジンコンソールを使用してワーカーノードを追加することができます。

- コンソールを使用したクラスターの作成

7.4. 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 つ以上の新規コンピュート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、元のコンピュート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。

7.4.1. コンピュート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デプロイメント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。

7.4.2. コンピュートマシンセットの手動スケーリング

コンピュート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、コンピュート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 installer-provisioned infrastructure のインストールに関連します。user-provisioned infrastructure のカスタマイズされたインストールにはコンピュート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. 次のコマンドを実行して、クラスター内のコンピュートマシンセットを表示します。

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

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

2. 次のコマンドを実行して、クラスター内のコンピュートマシンを表示します。

```
$ oc get machines.machine.openshift.io -n openshift-machine-api
```

3. 次のコマンドを実行して、削除するコンピュートマシンに注釈を設定します。

```
$ oc annotate machines.machine.openshift.io/<machine_name> -n openshift-machine-api
machine.openshift.io/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 machinesets.machine.openshift.io <machineset> -n openshift-
machine-api
```

   または、以下を実行します。

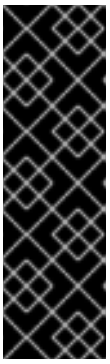
```
$ oc edit machinesets.machine.openshift.io <machineset> -n openshift-machine-api
```

ヒント

または、以下の YAML を適用してコンピュート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

- 次のコマンドを実行して、目的のマシンが削除されたことを確認します。

```
$ oc get machines.machine.openshift.io
```

7.4.3. コンピュート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** です。これは、コンピュートマシンセットのスケールダウン時にランダムなマシンが選択され、削除されることを意味します。削除ポリシーは、特定のコンピュート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/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、ワーカーのコンピュートマシンセットを **0** にスケーリングできません。

注記

カスタムのコンピュートマシンセットは、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コンピュート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。

7.4.4. クラスタースコープのデフォルトノードセレクターの作成

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レ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レ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レクターを Pod に追加し、一致するラベルのあるノードで Pod をスケジュールします。

スケジューラー Operator カスタムリソース (CR) を編集して、クラスタースコープのノードセレクターを設定します。ラベルをノード、コンピュートマシンセット、またはマシン設定に追加します。コンピュート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レ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
```

ノードセレ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レ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レクターは、Pod の再起動まで有効になりません。

2. コンピュート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。

   - コンピュートマシンセットを使用して、ノードの作成時にコンピュート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 を適用してコンピュート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、そのコンピュート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   v1.29.4
```

- ラベルをノードに直接追加します。

a. ノードの **Node** オブジェクトを編集します。

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

たとえば、ノードにラベルを付けるには、以下を実行します。

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

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"
```

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

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

以下に例を示します。

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

出力例

```
NAME                                        STATUS   ROLES    AGE   VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49   Ready    worker   17m   v1.29.4
```

7.5. ワーカーレイテンシープロファイルを使用したレイテンシーの高い環境でのクラスターの安定性の向上

クラスター管理者が遅延テストを実行してプラットフォームを検証した際に、遅延が大きい場合でも安定性を確保するために、クラスターの動作を調整する必要性が判明することがあります。クラスター管理者が変更する必要があるのは、ファイルに記録されている 1 つのパラメーターだけです。このパラメーターは、監視プロセスがステータスを読み取り、クラスターの健全性を解釈する方法に影響を与える 4 つのパラメーターを制御するものです。1 つのパラメーターのみを変更し、サポートしやすく簡単な方法でクラスターをチューニングできます。

Kubelet プロセスは、クラスターの健全性を監視する上での出発点です。**Kubelet** は、OpenShift Container Platform クラスター内のすべてのノードのステータス値を設定します。Kubernetes コントローラーマネージャー (**kube controller**) は、デフォルトで 10 秒ごとにステータス値を読み取ります。ノードのステータス値を読み取ることができない場合、設定期間が経過すると、**kube controller** とそのノードとの接続が失われます。デフォルトの動作は次のとおりです。

1. コントロールプレーン上のノードコントローラーが、ノードの健全性を **Unhealthy** に更新し、ノードの **Ready** 状態を `Unknown` とマークします。
2. この操作に応じて、スケジューラーはそのノードへの Pod のスケジューリングを停止します。
3. ノードライフサイクルコントローラーが、**NoExecute** effect を持つ **node.kubernetes.io/unreachable** テイントをノードに追加し、デフォルトでノード上のすべての Pod を 5 分後にエビクトするようにスケジュールします。

この動作は、ネットワークが遅延の問題を起こしやすい場合、特にネットワークエッジにノードがある場合に問題が発生する可能性があります。場合によっては、ネットワークの遅延が原因で、Kubernetes コントローラーマネージャーが正常なノードから更新を受信できないことがあります。**Kubelet** は、

ノードが正常であっても、ノードから Pod を削除します。

この問題を回避するには、**ワーカーレイテンシープロファイル** を使用して、**Kubelet** と Kubernetes コントローラーマネージャーがアクションを実行する前にステータスの更新を待機する頻度を調整できます。これらの調整により、コントロールプレーンとワーカーノード間のネットワーク遅延が最適でない場合に、クラスターが適切に動作するようになります。

これらのワーカーレイテンシープロファイルには、3 つのパラメーターセットが含まれています。パラメーターは、遅延の増加に対するクラスターの反応を制御するように、慎重に調整された値で事前定義されています。試験により手作業で最良の値を見つける必要はありません。

クラスターのインストール時、またはクラスターネットワークのレイテンシーの増加に気付いたときはいつでも、ワーカーレイテンシープロファイルを設定できます。

7.5.1. ワーカーレイテンシープロファイルについて

ワーカーレイテンシープロファイルは、4 つの異なるカテゴリーからなる慎重に調整されたパラメーターです。これらの値を実装する 4 つのパラメーターは、**node-status-update-frequency**、**node-monitor-grace-period**、**default-not-ready-toleration-seconds**、および **default-unreachable-toleration-seconds** です。これらのパラメーターにより、遅延の問題に対するクラスターの反応を制御できる値を使用できます。手作業で最適な値を決定する必要はありません。

重要

これらのパラメーターの手動設定はサポートされていません。パラメーター設定が正しくないと、クラスターの安定性に悪影響が及びます。

すべてのワーカーレイテンシープロファイルは、次のパラメーターを設定します。

node-status-update-frequency

kubelet がノードのステータスを API サーバーにポストする頻度を指定します。

node-monitor-grace-period

Kubernetes コントローラーマネージャーが、ノードを異常とマークし、**node.kubernetes.io/not-ready** または **node.kubernetes.io/unreachable** テイントをノードに追加する前に、kubelet からの更新を待機する時間を秒単位で指定します。

default-not-ready-toleration-seconds

ノードを異常とマークした後、Kube API Server Operator がそのノードから Pod を削除するまでに待機する時間を秒単位で指定します。

default-unreachable-toleration-seconds

ノードを到達不能とマークした後、Kube API Server Operator がそのノードから Pod を削除するまでに待機する時間を秒単位で指定します。

次の Operator は、ワーカーレイテンシープロファイルの変更を監視し、それに応じて対応します。

- Machine Config Operator (MCO) は、ワーカーノードの **node-status-update-frequency** パラメーターを更新します。
- Kubernetes コントローラーマネージャーは、コントロールプレーンノードの **node-monitor-grace-period** パラメーターを更新します。
- Kubernetes API Server Operator は、コントロールプレーンノードの **default-not-ready-toleration-seconds** および **default-unreachable-toleration-seconds** パラメーターを更新します。

ほとんどの場合はデフォルト設定が機能しますが、OpenShift Container Platform は、ネットワークで通常よりも高いレイテンシーが発生している状況に対して、他に 2 つのワーカーレイテンシープロファイルを提供します。次のセクションでは、3 つのワーカーレイテンシープロファイルについて説明します。

デフォルトのワーカーレイテンシープロファイル

Default プロファイルを使用すると、各 **Kubelet** が 10 秒ごとにステータスを更新します (**node-status-update-frequency**)。**Kube Controller Manager** は、**Kubelet** のステータスを 5 秒ごとにチェックします (**node-monitor-grace-period**)。
Kubernetes コントローラーマネージャーは、**Kubelet** が異常であると判断するまでに、**Kubelet** からのステータス更新を 40 秒待機します。ステータスが提供されない場合、Kubernetes コントローラーマネージャーは、ノードに **node.kubernetes.io/not-ready** または **node.kubernetes.io/unreachable** テイントのマークを付け、そのノードの Pod を削除します。

そのノードの Pod に **NoExecute** テイントがある場合、その Pod は **tolerationSeconds** に従って実行されます。Pod にテイントがない場合、その Pod は 300 秒以内に削除されます (**Kube API Server** の **default-not-ready-toleration-seconds** および **default-unreachable-toleration-seconds** 設定)。

プロファイル	コンポーネント	パラメーター	値
デフォルト	kubelet	**node-status-update-frequency**	10s
	Kubelet コントローラーマネージャー	**node-monitor-grace-period**	40s
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	300s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	300s

中規模のワーカーレイテンシープロファイル

ネットワークレイテンシーが通常の場合、**MediumUpdateAverageReaction** プロファイルを使用します。
MediumUpdateAverageReaction プロファイルは、kubelet の更新の頻度を 20 秒に減らし、Kubernetes コントローラーマネージャーがそれらの更新を待機する期間を 2 分に変更します。そのノード上の Pod の Pod 排除期間は 60 秒に短縮されます。Pod に **tolerationSeconds** パラメーターがある場合、エビクションはそのパラメーターで指定された期間待機します。

Kubernetes コントローラーマネージャーは、ノードが異常であると判断するまでに 2 分間待機します。別の 1 分間でエビクションプロセスが開始されます。

プロファイル	コンポーネント	パラメーター	値
MediumUpdateAverageReaction	kubelet	**node-status-update-frequency**	20s
	Kubelet コントローラーマネージャー	**node-monitor-grace-period**	2m
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	60s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	60s

ワーカーの低レイテンシープロファイル

ネットワーク遅延が非常に高い場合は、**LowUpdateSlowReaction** プロファイルを使用します。**LowUpdateSlowReaction** プロファイルは、kubelet の更新頻度を 1 分に減らし、Kubernetes コントローラーマネージャーがそれらの更新を待機する時間を 5 分に変更します。そのノード上の Pod の Pod 排除期間は 60 秒に短縮されます。Pod に **tolerationSeconds** パラメーターがある場合、エビクションはそのパラメーターで指定された期間待機します。

Kubernetes コントローラーマネージャーは、ノードが異常であると判断するまでに 5 分間待機します。別の 1 分間でエビクションプロセスが開始されます。

プロファイル	コンポーネント	パラメーター	値
LowUpdateSlowReaction	kubelet	**node-status-update-frequency**	1m
	Kubelet コントローラーマネージャー	**node-monitor-grace-period**	5m
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	60s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	60s

7.5.2. ワーカーレイテンシープロファイルの使用と変更

ネットワークの遅延に対処するためにワーカー遅延プロファイルを変更するには、**node.config** オブジェクトを編集してプロファイルの名前を追加します。遅延が増加または減少したときに、いつでもプロファイルを変更できます。

ワーカーレイテンシープロファイルは、一度に 1 つずつ移行する必要があります。たとえば、**Default** プロファイルから **LowUpdateSlowReaction** ワーカーレイテンシープロファイルに直接移行することはできません。まず **Default** ワーカーレイテンシープロファイルから **MediumUpdateAverageReaction** プロファイルに移行し、次に **LowUpdateSlowReaction** プロファイルに移行する必要があります。同様に、**Default** プロファイルに戻る場合は、まずロープロファイルからミディアムプロファイルに移行し、次に **Default** に移行する必要があります。

注記

OpenShift Container Platform クラスターのインストール時にワーカーレイテンシープロファイルを設定することもできます。

手順

デフォルトのワーカーレイテンシープロファイルから移動するには、以下を実行します。

1. 中規模のワーカーのレイテンシープロファイルに移動します。
   a. **node.config** オブジェクトを編集します。

```
$ oc edit nodes.config/cluster
```

   b. **spec.workerLatencyProfile: MediumUpdateAverageReaction** を追加します。

node.config オブジェクトの例

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
  - apiVersion: config.openshift.io/v1
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  workerLatencyProfile: MediumUpdateAverageReaction (1)

  # ...
```

(1) 中規模のワーカーレイテンシーポリシーを指定します。

変更が適用されると、各ワーカーノードでのスケジューリングは無効になります。

2. 必要に応じて、ワーカーのレイテンシーが低いプロファイルに移動します。

   a. **node.config** オブジェクトを編集します。

   ```
   $ oc edit nodes.config/cluster
   ```

   b. **spec.workerLatencyProfile** の値を **LowUpdateSlowReaction** に変更します。

   node.config オブジェクトの例

   ```
   apiVersion: config.openshift.io/v1
   kind: Node
   metadata:
     annotations:
       include.release.openshift.io/ibm-cloud-managed: "true"
       include.release.openshift.io/self-managed-high-availability: "true"
       include.release.openshift.io/single-node-developer: "true"
       release.openshift.io/create-only: "true"
     creationTimestamp: "2022-07-08T16:02:51Z"
     generation: 1
     name: cluster
     ownerReferences:
     - apiVersion: config.openshift.io/v1
       kind: ClusterVersion
       name: version
       uid: 36282574-bf9f-409e-a6cd-3032939293eb
     resourceVersion: "1865"
     uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
   spec:
     workerLatencyProfile: LowUpdateSlowReaction (1)

   # ...
   ```

   (1) 低ワーカーレイテンシーポリシーの使用を指定します。

変更が適用されると、各ワーカーノードでのスケジューリングは無効になります。

検証

- 全ノードが **Ready** 状態に戻ると、以下のコマンドを使用して Kubernetes Controller Manager を確認し、これが適用されていることを確認できます。

  ```
  $ oc get KubeControllerManager -o yaml | grep -i workerlatency -A 5 -B 5
  ```

  出力例

  ```
  # ...
      - lastTransitionTime: "2022-07-11T19:47:10Z"
        reason: ProfileUpdated
        status: "False"
        type: WorkerLatencyProfileProgressing
  ```

```
    - lastTransitionTime: "2022-07-11T19:47:10Z" (1)
      message: all static pod revision(s) have updated latency profile
      reason: ProfileUpdated
      status: "True"
      type: WorkerLatencyProfileComplete
    - lastTransitionTime: "2022-07-11T19:20:11Z"
      reason: AsExpected
      status: "False"
      type: WorkerLatencyProfileDegraded
    - lastTransitionTime: "2022-07-11T19:20:36Z"
      status: "False"
# ...
```

(1) プロファイルが適用され、アクティブであることを指定します。

ミディアムプロファイルからデフォルト、またはデフォルトからミディアムに変更する場合、**node.config** オブジェクトを編集し、**spec.workerLatencyProfile** パラメーターを適切な値に設定します。

7.6. コントロールプレーンマシンの管理

コントロール プレーンマシンセット は、コンピュートマシンセットがコンピュートマシンに提供するものと同様の管理機能をコントロールプレーンマシンに提供します。クラスター上のコントロールプレーンマシンセットの可用性と初期ステータスは、クラウドプロバイダーと、インストールした OpenShift Container Platform のバージョンによって異なります。詳細は、コントロールプレーンマシンセットの概要 を参照してください。

7.7. 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

コンピュートマシンセットを作成して、デフォルトのルーター、統合コンテナー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 つ以上のコンピュート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ー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ーゾーンに 1 つずつ、3 つの異なるコンピュートマシンセットが必要です。複数のアベイラビリティーゾーンを持たないグローバル Azure リージョンでは、アベイラビリティー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、マシンセットを使用する か ノードにラベルを割り当てる か、マシン設定プールを使用します。

この手順で使用することの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。

特定のノードセレ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。

7.7.1. コンピュート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. コンピュートマシンセットのカスタムリソース (CR) サンプルを含む新しい YAML ファイルを作成し、**<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ト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. 特定のコンピュートマシンセットカスタムリソース (CR) 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> 1
  name: <infrastructure_id>-<role> 2
  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: 3
      ...
```

1 クラスターインフラストラクチャー ID。

2 デフォルトのノードラベル。

注記

user-provisioned infrastructure を持つクラスターの場合、コンピュートマシンセットは **worker** および **infra** タイプのマシンのみを作成できます。

3 コンピュートマシンセット 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トマシンセット CR 設定を参照してください。

3. 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、コンピュートマシンセットのリストを表示します。

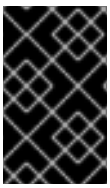
```
$ oc get machineset -n openshift-machine-api
```

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
```

新しいコンピュートマシンセットが利用可能になると、**DESIRED** と **CURRENT** の値が一致します。コンピュートマシンセットが使用できない場合は、数分待ってからコマンドを再実行してください。

7.7.2. 専用インフラストラクチャーノードの作成



重要

installer-provisioned infrastructure 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** とも呼ばれる) として指定できます。

手順

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

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

2. インフラストラクチャーノードとして機能する必要の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スコープのセレ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レクターはすべての namespace で作成された Pod に適用されます。これにより、Pod の既存のノードセレクターとの交差が作成され、Pod のセレクターをさらに制限します。

重要

デフォルトのノードセレクターのキーが 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レ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レ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レクターを **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レ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レクターを使用して、クラスター全体でのノードセレ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. 適切なノードセレクターと共に **defaultNodeSelector** フィールドを追加します。

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
spec:
  defaultNodeSelector: node-role.kubernetes.io/infra="" (1)
# ...
```

(1) この例のノードセレクターは、デフォルトでインフラストラクチャーノードに 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レクターを設定してクラスター全体のノードセレ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。

7.7.3. インフラストラクチャーマシンのマシン設定プール作成

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

重要

カスタムマシン設定プールを作成すると、デフォルトのワーカープール設定がオーバーライドされます (デフォルトのワーカープール設定が同じファイルまたはユニットを参照する場合)。

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レ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]} (1)
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/infra: "" (2)
```

(1) ワーカーロールおよびカスタムロールを追加します。

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

注記

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

3. YAML ファイルを用意した後に、マシン設定プールを作成できます。

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

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

```
$ oc get machineconfig
```

出力例

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   CREATED
00-master                                          365c1cfd14de5b0e3b85e0fc815b0060f36ab955
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 (1)
spec:
```

```
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - path: /etc/infratest
        mode: 0644
        contents:
          source: data:,infra
```

1 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。

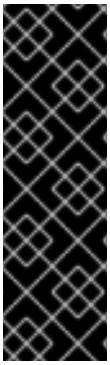
7.8. マシンセットリソースのインフラストラクチャーノードへの割り当て

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

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

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

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



重要

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

前提条件

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

手順

1. テイントを infra ノードに追加し、ユーザーのワークロードをこれにスケジュールできないようにします。

   a. ノードにテイントがあるかどうかを判別します。

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

   出力例

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

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

   b. ユーザーワークロードをスケジュー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: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: <node_name>
  labels:
    ...
spec:
  taints:
    - key: node-role.kubernetes.io/infra
      effect: NoSchedule
      value: reserved
  ...
```

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

注記

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

c. 上記の NoSchedule Effect のテイントとともに、NoExecute Effect のテイントを追加します。

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

以下に例を示します。

```
$ oc adm taint nodes node1 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、**node-role.kubernetes.io/infra** キーおよび **NoExecute** effect のテイントを持つ **node1** に配置します。**NoExecute** effect を持つノードは、テイントを容認する Pod のみをスケジュールします。effect は、一致する容認を持たないノードから既存の Pod を削除します。

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

```
tolerations:
  - effect: NoSchedule (1)
    key: node-role.kubernetes.io/infra (2)
    value: reserved (3)
  - effect: NoExecute (4)
    key: node-role.kubernetes.io/infra (5)
    operator: Exists (6)
    value: reserved (7)
```

(1) ノードに追加した effect を指定します。

(2) ノードに追加したキーを指定します。

(3) ノードに追加したキーと値のペア Taint の値を指定します。

(4) ノードに追加した effect を指定します。

(5) ノードに追加したキーを指定します。

(6) **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。

(7) 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。

7.9. 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

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

7.9.1. ルーターの移動

ルーター Pod を異なるコンピュート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

- 追加のコンピュートマシンセットを 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: 1
        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
```

1 適切な値が設定された **nodeSelector** パラメーターを、移動する必要の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 のリストを表示し、実行中の Pod のノード名をメモします。

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

   出力例

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

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

   b. 実行中の Pod のノードのステータスを表示します。

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

   1 Pod のリストより取得した **<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   v1.29.4
   ```

   ロールのリストに **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。

7.9.2. 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

- 追加のコンピュートマシンセットを OpenShift Container Platform クラスターに設定します。

手順

1. **config/instance** オブジェクトを表示します。

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

出力例

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

2. **config/instance** オブジェクトを編集します。

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

```
spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - podAffinityTerm:
          namespaces:
          - openshift-image-registry
          topologyKey: kubernetes.io/hostname
        weight: 100
  logLevel: Normal
  managementState: Managed
  nodeSelector: (1)
    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
```

1 適切な値が設定された **nodeSelector** パラメーターを、移動する必要の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** リストにあることを確認します。

7.9.3. 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、Thanos Querier、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** config map 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: 1
        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
  metricsServer:
    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
  telemeterClient:
    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
  monitoringPlugin:
    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、移動する必要の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。

7.9.4. ロギングリソースの移動

ロギングリソースの移動について、詳しくは以下を参照してください。

- ノードセレクターを使用したロギングリソースの移動
- テイントと容認を使用したロギング Pod の配置制御

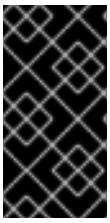
7.10. CLUSTER AUTOSCALER について

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 スコープがあり、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リングする可能性のあるコンピュート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レ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 をスケジュールできますが、これにより Cluster Autoscaler がリソースを増やすことはなく、余分なリソースがある場合にのみ実行されます。

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

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。

7.10.1. Cluster Autoscaler 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
  logVerbosity: 4 10
  scaleDown: 11
    enabled: true 12
    delayAfterAdd: 10m 13
    delayAfterDelete: 5m 14
    delayAfterFailure: 30s 15
    unneededTime: 5m 16
    utilizationThreshold: "0.4" 17
  expanders: ["Random"] 18
```

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トマシン、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. ロギングの詳細レベルを **0** から **10** の間で指定します。次のログレベルのしきい値は、ガイダンスとして提供されています。
    - **1**: (デフォルト) 変更に関する基本情報。
    - **4**: 一般的な問題をトラブルシューティングするためのデバッグレベルの詳細度。
    - **9**: 広範なプロトコルレベルのデバッグ情報。

値を指定しない場合は、デフォルト値の **1** が使用されます。

11 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。

12 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。

13 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。

14 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。

15 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。

16 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。

17 オプション: **node utilization level** を指定します。この使用率レベルを下回るノードは、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。

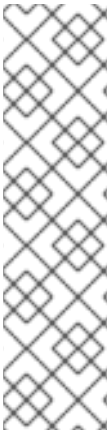
18 オプション: クラスターオートスケーラーで使用するエクスパンダーを指定します。次の値が有効です。

- **LeastWaste** : スケーリング後にアイドル CPU を最小限に抑えるマシンセットを選択します。複数のマシンセットで同じ量のアイドル CPU が生成される場合、選択によって未使用のメモリーが最小限に抑えられます。
- **Priority**: ユーザーが割り当てた優先度が最も高いマシンセットを選択します。このエクスパンダーを使用するには、マシンセットの優先順位を定義する config map を作成する必要があります。詳細は、「クラスターオートスケーラーの優先度エクスパンダーの設定」を参照してください。
- **Random**: (デフォルト) マシンセットをランダムに選択します。

値を指定しない場合は、デフォルト値 **Random** が使用されます。

[LeastWaste, Priority] 形式を使用して複数のエクスパンダーを指定できます。クラスターオートスケーラーは、指定された順序に従って各エクスパンダーを適用します。

[LeastWaste, Priority] の例では、クラスターオートスケーラーは最初に **LeastWaste** 基準に従って評価します。複数のマシンセットが **LeastWaste** 基準を同等に満たしている場合、クラスターオートスケーラーは **Priority** 基準に従って評価します。複数のマシンセットが指定されたエクスパンダーのすべてを同等に満たす場合、クラスターオートスケーラーはランダムに 1 つを選択して使用します。



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。

7.10.2. 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. カスタム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. 以下のコマンドを実行して、クラスター内にカスタムリソースを作成します。

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

1 **<filename>** はカスタムリソースファイルの名前です。

7.11. MACHINE AUTOSCALER について

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

重要

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

7.11.1. Machine Autoscaler 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" 1
```

```
  namespace: "openshift-machine-api"
spec:
  minReplicas: 1 2
  maxReplicas: 12 3
  scaleTargetRef: 4
    apiVersion: machine.openshift.io/v1beta1
    kind: MachineSet 5
    name: worker-us-east-1a 6
```

1 Machine Autoscaler の名前を指定します。この Machine Autoscaler がスケーリングするコンピュートマシンセットを簡単に特定できるようにするには、スケーリングするコンピュートマシンセットの名前を指定するか、これを組み込みます。コンピュートマシンセットの名前は、**<clusterid>-<machineset>-<region>** の形式を使用します。

2 Cluster Autoscaler がクラスターのスケーリングを開始した後に、指定されたゾーンに残っている必要のある指定されたタイプのマシンの最小数を指定します。AWS、GCP、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コンピュートマシンセットをスケーリングすることで、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コンピュート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 つのコンピュート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リ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。Machine Autoscaler がこの数のマシンをデプロイできるように、**ClusterAutoscaler** リソース定義の **maxNodesTotal** 値が十分に大きいことを確認してください。

4 このセクションでは、スケーリングする既存のコンピュート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コンピュートマシンセットの名前に一致する必要があります。

7.11.2. 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. カスタム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. 以下のコマンドを実行して、クラスター内にカスタムリソースを作成します。

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

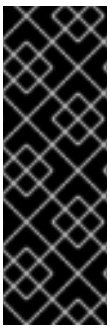
1 **<filename>** はカスタムリソースファイルの名前です。

7.12. LINUX CGROUP の設定

OpenShift Container Platform 4.14 以降、OpenShift Container Platform はクラスター内で Linux コントロールグループバージョン 2 (cgroup v2) を使用します。OpenShift Container Platform 4.13 以前で cgroup v1 を使用している場合、OpenShift Container Platform 4.14 以降に移行しても、cgroup 設定はバージョン 2 に自動的に更新されません。OpenShift Container Platform 4.14 以降の新規インストールでは、デフォルトで cgroup v2 が使用されます。ただし、インストール時に Linux コントロールグループバージョン 1 (cgroup v1) を有効にできます。

cgroup v2 は、Linux cgroup API の現行バージョンです。cgroup v2 では、統一された階層、安全なサブツリー委譲、Pressure Stall Information 等の新機能、および強化されたリソース管理および分離など、cgroup v1 に対していくつかの改善が行われています。ただし、cgroup v2 には、cgroup v1 とは異なる CPU、メモリー、および I/O 管理特性があります。したがって、一部のワークロードでは、cgroup v2 を実行するクラスター上のメモリーまたは CPU 使用率にわずかな違いが発生する可能性があります。

必要に応じて、cgroup v1 と cgroup v2 の間で変更できます。OpenShift Container Platform で cgroup v1 を有効にすると、クラスター内のすべての cgroup v2 コントローラーと階層が無効になります。



重要

cgroup v1 は非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

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。

注記

現在、CPU 負荷分散の無効化は cgroup v2 ではサポートされていません。その結果、cgroup v2 が有効になっている場合は、パフォーマンスプロファイルから望ましい動作が得られない可能性があります。パフォーマンスプロファイルを使用している場合は、cgroup v2 を有効にすることは推奨されません。

前提条件

- OpenShift Container Platform クラスター (バージョン 4.12 以降) が実行中。
- 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. ノードで cgroup v1 を有効にします。
   a. **node.config** オブジェクトを編集します。

```
$ oc edit nodes.config/cluster
```

b. Add **spec.cgroupMode: "v1"**:

node.config オブジェクトの例

```
apiVersion: config.openshift.io/v2
kind: Node
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
  - apiVersion: config.openshift.io/v2
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  cgroupMode: "v1" (1)
...
```

(1) cgroup v1 を有効にします。

検証

1. マシン設定をチェックして、新しいマシン設定が追加されたことを確認します。

```
$ oc get mc
```

出力例

```
NAME                                               GENERATEDBYCONTROLLER
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
97-master-generated-kubelet                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-worker-generated-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-23d4317815a5f854bd3553d689cfe2e9
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             10s (1)
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
rendered-worker-dcc7f1b92892d34db74d6832bcc9ccd4
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             10s
```

(1) 予想どおり、新しいマシン設定が作成されます。

2. 新しい **kernelArguments** が新しいマシン設定に追加されたことを確認します。

```
$ oc describe mc <name>
```

cgroup v1 の出力例

```
apiVersion: machineconfiguration.openshift.io/v2
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 05-worker-kernelarg-selinuxpermissive
spec:
  kernelArguments:
    systemd.unified_cgroup_hierarchy=0 (1)
    systemd.legacy_systemd_cgroup_controller=1 (2)
```

(1) cgroup v2 を無効にします。

(2) systemd で cgroup v1 を有効にします。

3. ノードをチェックして、ノードのスケジューリングが無効になっ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-fm1qnwt-72292-99kt6-master-0  Ready,SchedulingDisabled   master   58m
v1.29.4
ci-ln-fm1qnwt-72292-99kt6-master-1  Ready                      master   58m   v1.29.4
ci-ln-fm1qnwt-72292-99kt6-master-2  Ready                      master   58m   v1.29.4
ci-ln-fm1qnwt-72292-99kt6-worker-a-h5gt4   Ready,SchedulingDisabled   worker   48m
```

```
v1.29.4
ci-ln-fm1qnwt-72292-99kt6-worker-b-7vtmd   Ready                worker   48m   v1.29.4
ci-ln-fm1qnwt-72292-99kt6-worker-c-rhzkv   Ready                worker   48m   v1.29.4
```

4. ノードが **Ready** 状態に戻ったら、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>
```

5. **/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
```

6. **sys/fs/cgroup/cgroup2fs** ファイルがノードに存在することを確認します。このファイルは cgroup v1 によって作成されます。

```
$ stat -c %T -f /sys/fs/cgroup
```

出力例

```
cgroup2fs
```

関連情報

- ノードでの Linux cgroup バージョンの設定

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

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

7.13.1. 機能ゲートについて

FeatureGate カスタムリソース (CR) を使用して、クラスター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。この機能セットを使用すると、テストクラスターでこれらのテクノロジープレビュー機能を有効にすることができます。そこでは、これらの機能を完全にテストできますが、運用クラスターでは機能を無効にしたままにできます。

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。

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。

- 外部クラウドプロバイダー。vSphere、AWS、Azure、GCP 上にあるクラスターの外部クラウドプロバイダーのサポートを有効にします。OpenStack のサポートは GA です。これは内部機能であり、ほとんどのユーザーは操作する必要はありません。(**ExternalCloudProvider**)

- OpenShift Builds の共有リソース CSI ドライバー。Container Storage Interface (CSI) を有効にします。(**CSIDriverSharedResource**)

- ノード上のスワップメモリー。ノードごとに OpenShift Container Platform ワークロードのスワップメモリーの使用を有効にします。(**NodeSwap**)

- OpenStack Machine API プロバイダー。このゲートは効果がなく、今後のリリースでこの機能セットから削除される予定です。(**MachineAPIProviderOpenStack**)

- Insights Operator。**InsightsDataGather** CRD を有効にし、ユーザーがいくつかの Insights データ収集オプションを設定できるようにします。この機能セットにより、**DataGather** CRD も有効になり、ユーザーがオンデマンドで Insights データ収集を実行できるようになります。(**InsightsConfigAPI**)

- Retroactive デフォルトストレージクラス。PVC 作成時にデフォルトのストレージクラスがない場合に、OpenShift Container Platform は PVC に対してデフォルトのストレージクラスを遡及的に割り当てることができます。(**RetroactiveDefaultStorageClass**)

- 動的リソース割り当て API。Pod とコンテナー間でリソースを要求および共有するための新しい API が有効になります。これは内部機能であり、ほとんどのユーザーは操作する必要はありません。(**DynamicResourceAllocation**)

- Pod セキュリティーアドミッションの適用。Pod セキュリティーアドミッションの制限付き強制モードを有効にします。警告をログに記録するだけでなく、Pod のセキュリティー基準に違反している場合、Pod は拒否されます。(**OpenShiftPodSecurityAdmission**)

- StatefulSet Pod の可用性アップグレードの制限。ユーザーは、更新中に使用できないステートフルセット Pod の最大数を定義できるため、アプリケーションのダウンタイムが削減されます。(**MaxUnavailableStatefulSet**)

- 管理ネットワークポリシーとベースライン管理ネットワークポリシー。OVN-Kubernetes CNI プラグインを実行しているクラスターで、Network Policy V2 API に含まれる **AdminNetworkPolicy** リソースと **BaselineAdminNetworkPolicy** リソースを有効にします。クラスター管理者は、namespace が作成される前に、クラスター範囲のポリシーと保護措置をクラスター全体に適用できます。ネットワーク管理者は、ユーザーが上書きできないネットワークトラフィック制御を強制することで、クラスターを保護できます。ネットワーク管理者は、必要に応じて、クラスター内のユーザーが上書きできる任意のベースラインネットワークトラフィック制御を強制できます。現在、これらの API はクラスター内トラフィックのポリシーの表現のみをサポートしています。(**AdminNetworkPolicy**)

- **MatchConditions** は、この Webhook にリクエストを送信するために満たす必要がある条件のリストです。Match Conditions は、ルール、namespaceSelector、および objectSelector ですでに一致しているリクエストをフィルター処理します。**matchConditions** の空のリストは、すべてのリクエストに一致します。(**admissionWebhookMatchConditions**)

- **gcpLabelsTags**

- **vSphereStaticIPs**

- routeExternalCertificate
- automatedEtcdBackup
- gcpClusterHostedDNS
- vSphereControlPlaneMachineset
- dnsNameResolver
- machineConfigNodes
- metricsServer
- installAlternateInfrastructureAWS
- sdnLiveMigration
- mixedCPUsAllocation
- managedBootImages
- onClusterBuild
- signatureStores
- DisableKubeletCloudCredentialProviders
- BareMetalLoadBalancer
- ClusterAPIInstallAWS
- ClusterAPIInstallNutanix
- ClusterAPIInstallOpenStack
- ClusterAPIInstallVSphere
- HardwareSpeed
- KMSv1
- NetworkDiagnosticsConfig
- VSphereDriverConfiguration
- ExternalOIDC
- ChunkSizeMiB
- ClusterAPIInstallGCP
- ClusterAPIInstallPowerVS
- EtcdBackendQuota
- 例

- **ExternalRouteCertificate**
- **ImagePolicy**
- **InsightsConfig**
- **InsightsOnDemandDataGather**
- **MetricsCollectionProfiles**
- **NewOLM**
- **NodeDisruptionPolicy**
- **PinnedImages**
- **PlatformOperators**
- **ServiceAccountTokenNodeBinding**
- **ServiceAccountTokenNodeBindingValidation**
- **ServiceAccountTokenPodNodeInfo**
- **TranslateStreamCloseWebsocketRequests**
- **UpgradeStatus**
- **VSphereMultiVCenters**
- **VolumeGroupSnapshot**

7.13.2. 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** フィーチャーゲートをクリックし、**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。

フィーチャー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

ノードが準備完了状態に戻った後、ノード上の **kubelet.conf** ファイルを確認することで、フィーチャー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。

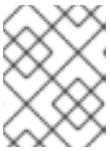
```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

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。

7.13.3. 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 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

ノードが準備完了状態に戻った後、ノード上の **kubelet.conf** ファイルを確認することで、フィーチャー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
```

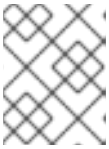
5. **kubelet.conf** ファイルを表示します。

```
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。

7.14. 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。

7.14.1. etcd 暗号化について

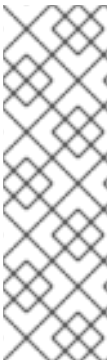
デフォルトで、etcd データは OpenShift Container Platform で暗号化されません。クラスターの etcd 暗号化を有効にして、データセキュリティーのレイヤーを追加で提供することができます。たとえば、etcd バックアップが正しくない公開先に公開される場合に機密データが失われないように保護することができます。

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。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_<datetimestamp>.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。セキュリティー上の理由から、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。

7.14.2. サポートされている暗号化の種類

以下の暗号化タイプは、OpenShift Container Platform で etcd データを暗号化するためにサポートされています。

AES-CBC

暗号化を実行するために、PKCS#7 パディングと 32 バイトの鍵を含む AES-CBC を使用します。暗号化キーは毎週ローテーションされます。

AES-GCM

AES-GCM とランダムナンスおよび 32 バイトキーを使用して暗号化を実行します。暗号化キーは毎週ローテーションされます。

7.14.3. 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。

etcd データベースは、AES-GCM または AES-CBC 暗号化で暗号化できます。

注記

etcd データベースをある暗号化タイプから別の暗号化タイプに移行するには、API サーバーの **spec.encryption.type** フィールドを変更します。etcd データの新しい暗号化タイプへの移行は自動的に行われます。

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. **spec.encryption.type** フィールドを **aesgcm** または **aescbc** に設定します。

```
spec:
  encryption:
    type: aesgcm (1)
```

(1) AES-CBC 暗号化の場合は **aescbc** に、AES-GCM 暗号化の場合は **aesgcm** に設定します。

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。etcd データベースのサイズによっては、このプロセスが完了するまでに 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。

7.14.4. etcd 暗号化の無効化

クラスターで etcd データの暗号化を無効にできます。

前提条件

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

手順

1. **APIServer** オブジェクトを変更します。

```
$ oc edit apiserver
```

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

```
spec:
  encryption:
    type: identity (1)
```

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

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

4. etcd の復号化が正常に行われたことを確認します。

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

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

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

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

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

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

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

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

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

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

   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。

7.14.5. 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ー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、プロキシーが有効にされているかどうかを確認できます。プロキシーは、**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 root としてデバッグセッションを開始します。

```
$ oc debug --as-root node/<node_name>
```

2. デバッグシェルで root ディレクトリーを **/host** に変更します。

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

3. クラスター全体のプロキシー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. デバッグシェルで **cluster-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
```

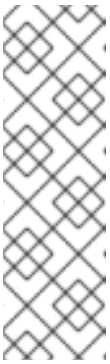
スクリプトの出力例

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



注記

etcd 暗号化が有効にされている場合、セキュリティー上の理由から、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。

7.14.6. 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。

7.14.6.1. 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>**

7.14.6.2. 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 (**(etcd_mvcc_db_total_size_in_bytes - 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 のリストを取得します。

```
$ 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 |   3.5.9 |  104 MB |     false |      false |
7 |      91624 |             91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.5.9 |  104 MB |     false |      false |
7 |      91624 |             91624 |        |
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.5.9 |  104 MB |      true |      false |
7 |      91624 |             91624 |        |
+---------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
```

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

2. etcd メンバーのデフラグ。

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

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

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

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

   c. etcd メンバーのデフラグを実行します。

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

   出力例

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

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

   d. データベースサイズが縮小されていることを確認します。

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

   出力例

```
+--------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
|        ENDPOINT          |        ID       | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+--------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
| https://10.0.191.37:2379 | 251cd44483d811c3 |   3.5.9 |  104 MB |     false |      false |
7 |      91624 |            91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.5.9 |   41 MB |     false |      false |
7 |      91624 |            91624 |        | (1)
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.5.9 |  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
```

7.14.7. 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。

注記

クラスターがコントロールプレーンマシンセットを使用している場合、より簡単な **etcd** リカバリー手順については、「コントロールプレーンマシンセットのトラブルシューティング」を参照してください。

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

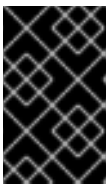
- インストール時に使用したものと同様、証明書ベースの **kubeconfig** ファイルを介して、**cluster-admin** ロールを持つユーザーとしてクラスターにアクセスします。
- リカバリーホストとして使用する正常なコントロールプレーンホストがあること。
- コントロールプレーンホストへの SSH アクセス。
- **etcd** スナップショットと静的 Pod のリソースの両方を含むバックアップディレクトリー (同じバックアップから取られるもの)。ディレクトリー内のファイル名は、**snapshot_<datetimestamp>.db** および **static_kuberesources_<datetimestamp>.tar.gz** の形式にする必要があります。

重要

非リカバリー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

1. リカバリーホストとして使用するコントロールプレーンホストを選択します。これは、復元操作を実行するホストです。
2. リカバリーホストを含む、各コントロールプレーンノードへの SSH 接続を確立します。**kube-apiserver** 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。

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

注記

リカバリーホストで静的 Pod を停止する必要はありません。

   a. リカバリーホストではないコントロールプレーンホストにアクセスします。

b. 以下を実行して、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 -v /etc/kubernetes/manifests/etcd-pod.yaml /tmp
```

c. 以下を使用して、**etcd** Pod が停止していることを確認します。

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

このコマンドの出力が空でない場合は、数分待ってからもう一度確認してください。

d. 以下を実行して、既存の **kube-apiserver** 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 -v /etc/kubernetes/manifests/kube-apiserver-pod.yaml /tmp
```

e. 以下を実行して、**kube-apiserver** コンテナーが停止していることを確認します。

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

このコマンドの出力が空でない場合は、数分待ってからもう一度確認してください。

f. 以下を使用して、既存の **kube-controller-manager** 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 -v /etc/kubernetes/manifests/kube-controller-manager-pod.yaml /tmp
```

g. 以下を実行して、**kube-controller-manager** コンテナーが停止していることを確認します。

```
$ sudo crictl ps | grep kube-controller-manager | egrep -v "operator|guard"
```

このコマンドの出力が空でない場合は、数分待ってからもう一度確認してください。

h. 以下を使用して、既存の **kube-scheduler** 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 -v /etc/kubernetes/manifests/kube-scheduler-pod.yaml /tmp
```

i. 以下を使用して、**kube-scheduler** コンテナーが停止していることを確認します。

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

このコマンドの出力が空でない場合は、数分待ってからもう一度確認してください。

j. 次の例を使用して、**etcd** データディレクトリーを別の場所に移動します。

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

k. **/etc/kubernetes/manifests/keepalived.yaml** ファイルが存在し、ノードが削除された場合は、次の手順に従います。

i. **/etc/kubernetes/manifests/keepalived.yaml** 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 -v /etc/kubernetes/manifests/keepalived.yaml /tmp
```

ii. **keepalived** デーモンによって管理されているコンテナーが停止していることを確認します。

```
$ sudo crictl ps --name keepalived
```

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。

iii. コントロールプレーンに仮想 IP (VIP) が割り当てられているかどうかを確認します。

```
$ ip -o address | egrep '<api_vip>|<ingress_vip>'
```

iv. 報告された仮想 IP ごとに、次のコマンドを実行して仮想 IP を削除します。

```
$ sudo ip address del <reported_vip> dev <reported_vip_device>
```

l. 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. **keepalived** デーモンが使用されている場合は、リカバリーコントロールプレーンノードが仮想 IP を所有していることを確認します。

```
$ ip -o address | grep <api_vip>
```

仮想 IP のアドレスが存在する場合、出力内で強調表示されます。仮想 IP が設定されていないか、正しく設定されていない場合、このコマンドは空の文字列を返します。

7. クラスター全体のプロキシー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、プロキシーが有効にされているかどうかを確認できます。プロキシーは、**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。

8. 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/assets/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
```

cluster-restore.sh スクリプトは、**etcd**、**kube-apiserver**、**kube-controller-manager**、および **kube-scheduler** Pod が停止され、復元プロセスの最後に開始されたことを示す必要があります。

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。

9. 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   v1.29.4
   host-172-25-75-38   Ready   infra,worker   3d20h   v1.29.4
   host-172-25-75-40   Ready   master        3d20h   v1.29.4
   host-172-25-75-65   Ready   master        3d20h   v1.29.4
   host-172-25-75-74   Ready   infra,worker   3d20h   v1.29.4
   host-172-25-75-79   Ready   worker        3d20h   v1.29.4
   host-172-25-75-86   Ready   worker        3d20h   v1.29.4
   host-172-25-75-98   Ready   infra,worker   3d20h   v1.29.4
   ```

   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
```

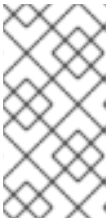
10. すべてのコントロールプレーンホストで kubelet サービスを再起動します。

    a. 復元ホストから以下を実行します。

    ```
    $ 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。

11. 保留中の証明書署名要求 (CSR) を承認します。



注記

単一ノードクラスターや 3 つのスケジュール可能なコントロールプレーンノードで設定されるクラスターなど、ワーカーノードを持たないクラスターには、承認する保留中の CSR はありません。この手順にリストされているすべてのコマンドをスキップできます。

    a. 次のコマンドを実行して、現在の CSR のリストを取得します。

    ```
    $ oc get csr
    ```

    出力例

    ```
    NAME        AGE    SIGNERNAME                                    REQUESTOR
    CONDITION
    csr-2s94x   8m3s   kubernetes.io/kubelet-serving                 system:node:<node_name>
    Pending (1)
    csr-4bd6t   8m3s   kubernetes.io/kubelet-serving                 system:node:<node_name>
    Pending (2)
    csr-4hl85   13m    kubernetes.io/kube-apiserver-client-kubelet
    system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
    (3)
    csr-zhhhp   3m8s   kubernetes.io/kube-apiserver-client-kubelet
    system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
    (4)
    ...
    ```

(1) (2) kubelet 提供エンドポイントのノードによって要求される、保留中の kubelet 提供 CSR。

(3) (4) **node-bootstrapper** ノードのブートストラップ認証情報を使用して要求される、保留中の kubelet クライアント CSR。

b. 次のコマンドを実行して、CSR の詳細と CSR が有効であることを確認します。

```
$ oc describe csr <csr_name> 1
```

1 **<csr_name>** は、現行の CSR のリストからの CSR の名前です。

c. 以下を実行して、有効な **node-bootstrapper** CSR をそれぞれ承認します。

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

d. user-provisioned installation の場合、以下を実行して各 kubelet service CSR を承認します。

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

12. 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. 以下を使用して、リカバリーホストから **etcd** コンテナー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
36f86e2eeaaffe662df0d21041eb22b8198e0e58abeeae8c743c3e6e977e8009
About a minute ago   Running             etcd                                0
7c05f8af362f0
```

b. 以下を使用して、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。

13. **OVNKubernetes** ネットワークプラグインを使用している場合は、**ovnkube-controlplane** Pod を再起動する必要があります。

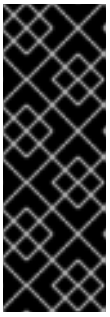
a. 以下を実行して、すべての **ovnkube-controlplane** Pod を削除します。

```
$ oc -n openshift-ovn-kubernetes delete pod -l app=ovnkube-control-plane
```

b. 次のコマンドを使用して、すべての **ovnkube-controlplane** Pod が再デプロイされたことを確認します。

```
$ oc -n openshift-ovn-kubernetes get pod -l app=ovnkube-control-plane
```

14. OVN-Kubernetes ネットワークプラグインを使用している場合は、すべてのノードで Open Virtual Network (OVN) Kubernetes Pod を 1 つずつ再起動します。次の手順を使用して、各ノードで OVN-Kubernetes Pod を再起動します。



重要

次の順序で OVN-Kubernetes Pod を再起動します。

1. リカバリーコントロールプレーンホスト
2. 他のコントロールプレーンホスト (利用可能な場合)
3. 他のノード

注記

検証および変更用の受付 Webhook は Pod を拒否することができます。**failurePolicy** を **Fail** に設定して追加の Webhook を追加すると、Pod が拒否され、復元プロセスが失敗する可能性があります。これは、クラスターの状態の復元中に Webhook を保存および削除することで回避できます。クラスターの状態が正常に復元された後に、Webhook を再度有効にできます。

または、クラスターの状態の復元中に **failurePolicy** を一時的に **Ignore** に設定できます。クラスターの状態が正常に復元された後に、**failurePolicy** を **Fail** にすることができます。

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) を使用して復元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、以下を実行します。

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

b. OpenVSwitch サービスを再起動します。Secure Shell (SSH) を使用してノードにアクセスし、次のコマンドを実行します。

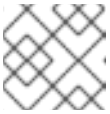
```
$ sudo systemctl restart ovs-vswitchd ovsdb-server
```

c. 次のコマンドを実行して、ノード上の **ovnkube-node** Pod を削除します。**<node>** は、再起動するノードの名前に置き換えます。

```
$ oc -n openshift-ovn-kubernetes delete pod -l app=ovnkube-node --field-
selector=spec.nodeName==<node>
```

d. 以下を使用して、**ovnkube-node** Pod が再度実行されていることを確認します。

```
$ oc -n openshift-ovn-kubernetes get pod -l app=ovnkube-node --field-
selector=spec.nodeName==<node>
```



注記

Pod が再起動するまでに数分かかる場合があります。

15. 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。

- installer-provisioned infrastructure 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。
>
> installer-provisioned infrastructure 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                                PHASE     TYPE        REGION      ZONE         AGE
NODE                           PROVIDERID                          STATE
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. 以下を実行して、マシン設定をファイルシステム上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. 以下を実行して、**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. 以下を実行して、**metadata.name** フィールドを新しい名前に変更します。

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. 以下を実行して、**spec.providerID** フィールドを削除します。

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

iv. 以下を実行して、**metadata.annotations** フィールドと **metadata.generation** フィールドを削除します。

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

v. 以下を実行して、**metadata.resourceVersion** フィールドと **metadata.uid** フィールドを削除します。

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

d. 以下を実行して、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 1
```

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。

e. 以下を実行して、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. 以下を実行して、**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. 以下を実行して、新しい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         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-master-3          Provisioning   m4.xlarge   us-east-1   us-east-1a   85s     ip-10-0-173-171.ec2.internal   aws:///us-east-1a/i-015b0888fe17bc2c8   running (1)
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) 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。

16. 次のように入力して、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。

17. リカバリーホスト内の別のターミナルウィンドウで、以下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
```

18. **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、以下を実行します。

```
$ 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。

19. 次のように入力して、クォーラムガードをオンに戻します。

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

20. 以下を実行すると、**unsupportedConfigOverrides** セクションがオブジェクトから削除されたことを確認できます。

```
$ oc get etcd/cluster -oyaml
```

21. すべてのノードが最新のリビジョンに更新されていることを確認します。
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** ステータス条件を確認し、すべてのノードが最新のリビジョンであることを確認します。更新が正常に実行されると、この出力には **AllNodesAtLatestRevision** が表示されます。

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

(1) この例では、最新のリビジョン番号は **7** です。

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

22. **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 は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、**kube-apiserver** 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を実行します。

    a. **kube-apiserver** の新規ロールアウトを強制します。

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

    すべてのノードが最新のリビジョンに更新されていることを確認します。

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

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

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

    (1) この例では、最新のリビジョン番号は **7** です。

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

    b. 次のコマンドを実行して、Kubernetes コントローラーマネージャーの新規ロールアウトを強制します。

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

    以下を実行して、すべてのノードが最新リビジョンに更新されていることを確認します。

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

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

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

    (1) この例では、最新のリビジョン番号は **7** です。

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

c. 以下を実行して、**kube-scheduler** の新規ロールアウトを強制します。

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

以下を使用して、すべてのノードが最新のリビジョンに更新されていることを確認します。

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

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

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

(1) この例では、最新のリビジョン番号は **7** です。

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

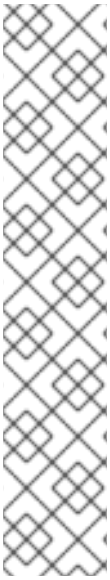
23. すべてのコントロールプレーンホストが起動しており、クラスターに参加していることを確認します。
クラスターにアクセスできるターミナルで、**cluster-admin** ユーザーとして以下のコマンドを実行します。

```
$ oc -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するには、**kube-apiserver** 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ー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

7.14.8. 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 クラスターがいずれかの形式の永続ストレージを使用する場合に、クラスターの状態は通常 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があります。

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 クラスターを復元する時に、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)。

7.15. 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。

7.15.1. 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、次の主要な部分で構成されます。

- 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レクター。

- 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。

警告

OpenShift Container Platform のすべてのマシン設定プールにおける **maxUnavailable** のデフォルト設定は **1** です。この値を変更せず、一度に 1 つのコントロールプレーンノードを更新することを推奨します。コントロールプレーンプールのこの値を **3** に変更しないでください。

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
```

注記

次の例には、OpenShift Container Platform on AWS に固有の値がいくつか含まれています。

出力例

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

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

注記

Pod の優先順位およびプリエンプシ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。

7.15.2. 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>
```

7.15.3. 正常でない Pod のエビクションポリシーの指定

Pod の Disruption Budget (PDB: 停止状態の予算) を使用して同時に利用可能にする必要のある Pod 数を指定する場合、正常でない Pod がエビクション対象とみなされる条件も定義できます。

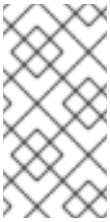
以下のポリシーから選択できます。

IfHealthyBudget

正常ではない実行中の Pod は、保護されたアプリケーションが停止されない場合に限りエビクトできます。

AlwaysAllow

正常ではない実行中の Pod は、Pod の Disruption Budget の条件が満たされているかどうかにかかわらずエビクトできます。このポリシーは、Pod が **CrashLoopBackOff** 状態でスタックしているアプリケーションや **Ready** ステータスの報告に失敗しているアプリケーションなど、正常に動作しないアプリケーションをエビクトするために使用できます。



注記

ノードドレイン中に誤動作するアプリケーションのエビクションをサポートするには、**PodDisruptionBudget** オブジェクトの **unhealthyPodEvictionPolicy** フィールドを **AlwaysAllow** に設定することを推奨します。デフォルトの動作では、ドレインを続行する前に、アプリケーション Pod が正常になるまで待機します。

手順

1. **PodDisruptionBudget** オブジェクトを定義する YAML ファイルを作成し、正常でない Pod のエビクションポリシーを指定します。

   pod-disruption-budget.yaml ファイルの例

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      name: my-pod
  unhealthyPodEvictionPolicy: AlwaysAllow 1
```

   1 正常でない Pod エビクションポリシーとして **IfHealthyBudget** または **AlwaysAllow** のいずれかを選択します。**unhealthyPodEvictionPolicy** フィールドが空の場合、デフォルトは **IfHealthyBudget** です。

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

```
$ oc create -f pod-disruption-budget.yaml
```

PDB で正常でない Pod のエビクションポリシーが **AlwaysAllow** に設定されている場合、ノードをドレイン (解放) 、この PDB が保護する正常に動作しないアプリケーションの Pod をエビクトできます。

関連情報

- フィーチャーゲートを使用した機能の有効化
- Kubernetes ドキュメントの Unhealthy Pod Eviction Policy

7.16. 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。

7.16.1. 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。場合によっては、Cluster Samples Operator 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ります。

この config map は、イメージストリームをインポートするためにミラーリングする必要があるイメージ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 config map 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** リストに、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。

7.16.2. 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。

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、非接続環境では **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/rhscl/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhscl/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。

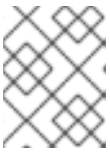
```
$ oc create configmap registry-config --from-
file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

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
```

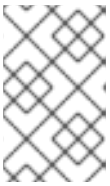


注記

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で必要になります。

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。

7.16.3. 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

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

   a. クラスターのイメージ設定オブジェクトを作成します。

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

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

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

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

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

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

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

7.17. 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

1. 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 -nopenshift
```

2. 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}" -nopenshift
```

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}" -nopenshift
```

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8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。

8.1. RHEL コンピュートマシンの OPENSHIFT CONTAINER PLATFORM クラスターへの追加

RHEL コンピュートノードについて理解し、これを使用します。

8.1.1. RHEL コンピュートノードのクラスターへの追加について

OpenShift Container Platform 4.16 では、**x86_64** アーキテクチャー上で user-provisioned infrastructure インストールまたは installer-provisioned infrastructure 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トマシンとして 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 コンピュート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。

installer-provisioned infrastructure クラスターの場合、installer-provisioned infrastructure 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 コンピュートマシンは、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。

8.1.2. RHEL コンピュートノードのシステム要件

OpenShift Container Platform 環境の Red Hat Enterprise Linux (RHEL) コンピュート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ト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、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 で実行されるインフ

- 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: "最小" インストールオプションを備えた RHEL 8.6 以降。

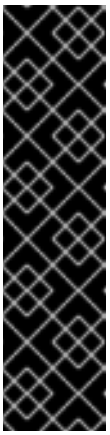
重要

OpenShift Container Platform クラスターへの RHEL 7 コンピュート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 コンピュートマシンがある場合、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り、古い RHEL 7 ホストを削除する必要があります。詳細は、「ノードの削除」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。

FIPS モードでブートされた Red Hat Enterprise Linux (RHEL) または Red Hat Enterprise Linux CoreOS (RHCOS) を実行する場合、OpenShift Container Platform コアコンポーネントは、x86_64、ppc64le、および s390x アーキテクチャーのみで、FIPS 140-2/140-3 検証のために NIST に提出された RHEL 暗号化ライブラリーを使用します。

- NetworkManager 1.0 以降。
- 1 vCPU。
- 最小 8 GB の RAM。
- **/var/** を含むファイルシステムの最小 15 GB のハードディスク領域。
- **/usr/local/bin/** を含むファイルシステムの最小 1 GB のハードディスク領域。
- 一時ディレクトリーを含むファイルシステムの最小 1 GB のハードディスク領域。システムの一時ディレクトリーは、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できる必要があります。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。

関連情報

- ノードの削除

8.1.2.1. 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。

8.1.3. Playbook 実行のためのマシンの準備

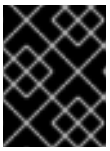
Red Hat Enterprise Linux (RHEL) をオペレーティングシステムとして使用するコンピュートマシンを OpenShift Container Platform 4.16 クラスターに追加する前に、新たなノードをクラスターに追加する Ansible Playbook を実行する RHEL 8 マシンを準備する必要があります。このマシンはクラスターの一部にはなりませんが、クラスターにアクセスできる必要があります。

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 8 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、コンピュート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 プロキシーまたは 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リスト表示します。

```
# 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 4.16 で必要なリポジトリーを有効にします。

```
# subscription-manager repos \
    --enable="rhel-8-for-x86_64-baseos-rpms" \
    --enable="rhel-8-for-x86_64-appstream-rpms" \
    --enable="rhocp-4.16-for-rhel-8-x86_64-rpms"
```

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

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

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

8.1.4. 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、必要なリポジトリー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リスト表示します。

```
# 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 リポジトリーをすべて無効にします。

   a. 有効にされている RHSM リポジトリーをすべて無効にします。

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

   b. 残りの yum リポジトリーをリスト表示し、**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 リポジトリーを無効にします。

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

   または、すべてのリポジトリーを無効にします。

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

   利用可能なリポジトリーが多い場合には、数分の時間がかかることがあります。

6. OpenShift Container Platform 4.16 で必要なリポジトリー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-4.16-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。

8.1.5. RHEL コンピュート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トマシンを、OpenShift Container Platform 4.16 クラスターに追加できます。

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. コンピュート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 (1)
#ansible_become=True (2)

openshift_kubeconfig_path="~/.kube/config" (3)

[new_workers] (4)
mycluster-rhel8-0.example.com
mycluster-rhel8-1.example.com
```

(1) Ansible タスクをリモートコンピュートマシンで実行するユーザー名を指定します。

(2) **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。

(3) クラスターの **kubeconfig** ファイルへのパスを指定します。

(4) クラスターに追加する各 RHEL マシンをリスト表示します。各ホストについて完全修飾ドメイン名を指定する必要があります。この名前は、クラスターがマシンにアクセスするために使用するホスト名であるため、マシンにアクセスできるように正しいパブリックまたはプライベートの名前を設定します。

2. Ansible Playbook ディレクトリーに移動します。

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

3. Playbook を実行します。

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

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

8.1.6. Ansible ホストファイルの必須パラメーター

Red Hat Enterprise Linux (RHEL) コンピュート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。

パラメーター	説明	値
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、**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。

8.1.7. オプション: RHCOS コンピュートマシンのクラスターからの削除

Red Hat Enterprise Linux (RHEL) コンピュート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) コンピュートマシンを削除し、リソースを解放できます。

前提条件

- RHEL コンピュートマシンをクラスターに追加済みです。

手順

1. マシンのリストを表示し、RHCOS コンピュー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 コンピュートマシンについて、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> 1
```

   1 RHCOS コンピュート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 1
```

   1 分離した RHCOS コンピュート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 コンピュートマシンのノード名を指定します。

3. コンピュートマシンのリストを確認し、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ト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 コンピュート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。

8.2. RHCOS コンピュートマシンの OPENSHIFT 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) コンピュート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ト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。

8.2.1. 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。

8.2.2. ISO イメージを使用した 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) コンピュートマシンを作成できます。

前提条件

- クラスターのコンピュートマシンの Ignition 設定ファイルの URL を取得します。このファイルがインストール時に HTTP サーバーにアップロードされている必要があります。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 次のコマンドを実行して、クラスターから Ignition 設定ファイルを抽出します。

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --to=- > worker.ign
```

2. クラスターからエクスポートした **worker.ign** Ignition 設定ファイルを HTTP サーバーにアップロードします。これらのファイルの URL をメモします。
3. Ignition ファイルが URL で利用可能であることを検証できます。次の例では、コンピュートノードの Ignition 設定ファイルを取得します。

```
$ curl -k http://<HTTP_server>/worker.ign
```

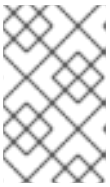
4. 次のコマンドを実行すると、新しいマシンを起動するための ISO イメージにアクセスできます。

```
RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get
configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.
<architecture>.artifacts.metal.formats.iso.disk.location')
```

5. ISO ファイルを使用して、追加のコンピュートマシンに RHCOS をインストールします。クラスターのインストール前にマシンを作成する際に使用したのと同じ方法を使用します。

   - ディスクに ISO イメージを書き込み、これを直接起動します。
   - LOM インターフェイスで ISO リダイレクトを使用します。

6. 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。

7. **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> 1 2
```

1. **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。
2. **--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。

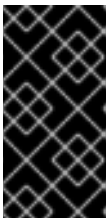
注記

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場合は、**coreos-installer** を実行する前に、内部認証局 (CA) をシステムのトラスト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
```

8. 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。

9. 継続してクラスター用の追加のコンピュートマシンを作成します。

8.2.3. PXE または iPXE ブートによる RHCOS マシンの作成

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

前提条件

- クラスターのコンピュートマシンの Ignition 設定ファイルの URL を取得します。このファイルがインストール時に HTTP サーバーにアップロードされている必要があります。
- クラスターのインストール時に HTTP サーバーにアップロードした RHCOS ISO イメージ、圧縮されたメタル BIOS、**kernel**、および **initramfs** ファイルの URL を取得します。
- インストール時に OpenShift Container Platform クラスターのマシンを作成するために使用した PXE ブートインフラストラクチャーにアクセスできる必要があります。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

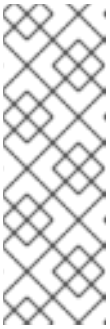
1. RHCOS イメージの PXE または iPXE インストールが正常に行われていることを確認します。
   - PXE の場合:

```
DEFAULT pxeboot
TIMEOUT 20
PROMPT 0
LABEL pxeboot
    KERNEL http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> (1)
    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 (2)
```

(1) 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。

(2) 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。



注記

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

- iPXE (**x86_**64 + **aarch64**):

```
kernel http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> initrd=main
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign (1) (2)
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img (3)
boot
```

(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.live.rootfs_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.inst.ignition_url** パラメーター値は **rootfs** のライブファイルの場所です。

(2) 複数の NIC を使用する場合、**ip** オプションに単一インターフェイスを指定します。たとえば、**eno1** という名前の NIC で DHCP を使用するには、**ip=eno1:dhcp** を設定します。

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

注記

この設定では、グラフィカルコンソールを備えたマシンでのシリアルコンソールアクセスは有効になりません。別のコンソールを設定するには、**kernel** 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? と、「高度な RHCOS インストール設定」セクションの「PXE および ISO インストール用シリアルコンソールの有効化」を参照してください。

注記

aarch64 アーキテクチャーで CoreOS **kernel** をネットワークブートするには、**IMAGE_GZIP** オプションが有効になっているバージョンの iPXE ビルドを使用する必要があります。iPXE の **IMAGE_GZIP** オプション を参照してください。

- **aarch64** 上の PXE (第 2 段階として UEFI および GRUB を使用) の場合:

```
menuentry 'Install CoreOS' {
    linux rhcos-<version>-live-kernel-<architecture>
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign 1 2
    initrd rhcos-<version>-live-initramfs.<architecture>.img 3
}
```

1. HTTP/TFTP 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、TFTP サーバー上の **kernel** ファイルの場所になります。**coreos.live.rootfs_url** パラメーター値は **rootfs** ファイルの場所であり、**coreos.inst.ignition_url** パラメーター値は HTTP サーバー上のブートストラップ Ignition 設定ファイルの場所になります。
2. 複数の NIC を使用する場合、**ip** オプションに単一インターフェイスを指定します。たとえば、**eno1** という名前の NIC で DHCP を使用するには、 **ip=eno1:dhcp** を設定します。
3. TFTP 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。

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

8.2.4. 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  v1.29.4
```

```
master-1  Ready    master  63m  v1.29.4
master-2  Ready    master  64m  v1.29.4
```

出力には作成したすべてのマシンがリスト表示されます。

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。

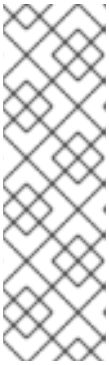
```
$ oc get csr
```

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。このリスト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 user-provisioned infrastructure 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 のリストからの 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 のリストからの 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。

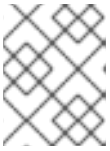
```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs oc adm certificate approve
```

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  v1.29.4
master-1  Ready     master  73m  v1.29.4
master-2  Ready     master  74m  v1.29.4
worker-0  Ready     worker  11m  v1.29.4
worker-1  Ready     worker  11m  v1.29.4
```



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。

8.2.5. 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) コンピュート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、新規コンピュートマシンセットを設定します。これらの手順は AWS クラスターに固有のものです。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", (1)
        "partitions": [
          {
            "label": "var",
            "sizeMiB": 50000, (2)
            "startMiB": 0 (3)
          }
        ]
      }
    ],
    "filesystems": [
      {
        "device": "/dev/disk/by-partlabel/var", (4)
        "format": "xfs", (5)
        "path": "/var" (6)
      }
    ]
  },
  "systemd": {
    "units": [ (7)
      {
        "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"
      }
```

```
    ]
  }
}
```

1. AWS ブロックデバイスへの絶対パスを指定します。
2. 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。
3. 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ブートディスクに追加する場合は、最小値の 25000 MB (メビバイト) が推奨されます。ルートファイルシステムは、指定したオフセットまでの利用可能な領域をすべて埋めるためにサイズを自動的に変更します。値の指定がない場合や、指定した値が推奨される最小値よりも小さい場合、生成されるルートファイルシステムのサイズは小さ過ぎるため、RHCOS の再インストールでデータパーティションの最初の部分が上書きされる可能性があります。
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実行される、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. 新規ノードの新規コンピュート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コンピュートマシンセット 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

既存のコンピュート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 (2)
            ebs:
              encrypted: true
              iops: 0
              volumeSize: 120
              volumeType: gp2
          - DeviceName: /dev/nvme1n2 (3)
            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 4
```

1. 新規ノードの名前を指定します。
2. 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)。
3. オプション: 追加の EBS ボリュームを指定します。
4. ユーザーデータシークレットファイルを指定します。

b. コンピュート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. コンピュート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-bdgsc-worker-us-east-1a   1         1         1       1           124m
ci-ln-2675bt2-76ef8-bdgsc-worker-us-east-1b   2         2         2       2           124m
worker-us-east-2-nvme1n1             1         1         1       1           2m35s 1
```

1. これが新しいコンピュートマシンセットです。

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    v1.29.4
ip-10-0-146-113.ec2.internal   Ready    master   127m    v1.29.4
ip-10-0-153-35.ec2.internal    Ready    worker   118m    v1.29.4
ip-10-0-176-58.ec2.internal    Ready    master   126m    v1.29.4
ip-10-0-217-135.ec2.internal   Ready    worker   2m57s   v1.29.4 1
ip-10-0-225-248.ec2.internal   Ready    master   127m    v1.29.4
ip-10-0-245-59.ec2.internal    Ready    worker   116m    v1.29.4
```

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 1
```

1 **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。

8.3. 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

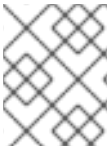
高度なマシン管理およびスケーリング機能は、マシン API が機能しているクラスターでのみ使用することができます。user-provisioned infrastructure を持つクラスターでは、Machine API を使用するために追加の検証と設定が必要です。

インフラストラクチャープラットフォームタイプが **none** のクラスターでは、Machine 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}'
```

8.3.1. マシンのヘルスチェック



注記

マシンのヘルスチェックは、コンピュートマシンセットまたはコントロールプレーンマシンセットにより管理されるマシンにのみ適用できます。

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。

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。

8.3.1.1. 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。
- 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。

関連情報

- コントロールプレーンマシンセットについて

8.3.2. 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 (1)
  namespace: openshift-machine-api
spec:
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-machine-role: <role> (2)
      machine.openshift.io/cluster-api-machine-type: <role> (3)
      machine.openshift.io/cluster-api-machineset: <cluster_name>-<label>-<zone> (4)
  unhealthyConditions:
  - type:    "Ready"
    timeout: "300s" (5)
    status: "False"
  - type:    "Ready"
    timeout: "300s" (6)
    status: "Unknown"
  maxUnhealthy: "40%" (7)
  nodeStartupTimeout: "10m" (8)
```

(1) デプロイするマシンヘルスチェックの名前を指定します。

(2)(3) チェックする必要のあるマシンプールのラベルを指定します。

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

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

(7) ターゲットプールで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。

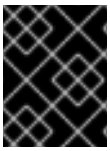
注記

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。

8.3.2.1. 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。

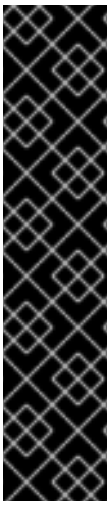
ユーザーがマシンの修復前に **maxUnhealthy** フィールドの値を定義する場合、**MachineHealthCheck** は **maxUnhealthy** の値を、正常でないと判別するターゲットプール内のマシン数と比較します。正常でないマシンの数が **maxUnhealthy** の制限を超える場合、修復は実行されません。



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ーゾーン間で複数のマシンセットに対応でき、ゾーン全体が失われると、**maxUnhealthy** の設定によりクラスター内で追加の修復を防ぐことができます。複数のアベイラビリティーゾーンを持たないグローバル Azure リージョンでは、アベイラビリティーセットを使用して高可用性を確保できます。



重要

コントロールプレーンの **MachineHealthCheck** リソースを設定する場合は、**maxUnhealthy** の値を **1** に設定します。

この設定により、複数のコントロールプレーンマシンが異常であると思われる場合に、マシンのヘルスチェックがアクションを実行しないことが保証されます。複数の異常なコントロールプレーンマシンは、etcd クラスターが劣化していること、または障害が発生したマシンを置き換えるためのスケーリング操作が進行中であることを示している可能性があります。

etcd クラスターが劣化している場合は、手動での介入が必要になる場合があります。スケーリング操作が進行中の場合は、マシンのヘルスチェックで完了できるようにする必要があります。

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。

8.3.2.1.1. 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。

8.3.2.1.2. 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:

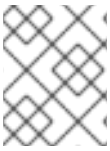
- 2 つ以下のノードが正常でない場合に、修復が実行されます。
- 3 つ以上のノードが正常でない場合は、修復は実行されません。

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。

8.3.3. マシンヘルスチェックリソースの作成

クラスター内のマシンセットの **MachineHealthCheck** リソースを作成できます。



注記

マシンのヘルスチェックは、コンピュートマシンセットまたはコントロールプレーンマシンセットにより管理されるマシンにのみ適用できます。

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
```

8.3.4. コンピュートマシンセットの手動スケーリング

コンピュート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、コンピュート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 installer-provisioned infrastructure のインストールに関連します。user-provisioned infrastructure のカスタマイズされたインストールにはコンピュート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. 次のコマンドを実行して、クラスター内のコンピュートマシンセットを表示します。

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

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

2. 次のコマンドを実行して、クラスター内のコンピュートマシンを表示します。

```
$ oc get machines.machine.openshift.io -n openshift-machine-api
```

3. 次のコマンドを実行して、削除するコンピュートマシンに注釈を設定します。

```
$ oc annotate machines.machine.openshift.io/<machine_name> -n openshift-machine-api
machine.openshift.io/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 machinesets.machine.openshift.io <machineset> -n openshift-
machine-api
```

または、以下を実行します。

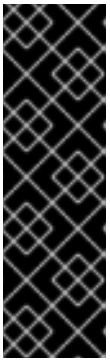
```
$ oc edit machinesets.machine.openshift.io <machineset> -n openshift-machine-api
```

ヒント

または、以下の YAML を適用してコンピュート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

- 次のコマンドを実行して、目的のマシンが削除されたことを確認します。

```
$ oc get machines.machine.openshift.io
```

8.3.5. コンピュート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デプロイメント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。

8.4. 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、**podsPerCore** および **maxPods** の 2 つのパラメーターはノードにスケジュールできる Pod の最大数を制御します。

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

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

重要

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

注記

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

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

```
kubeletConfig:
  podsPerCore: 10
```

podsPerCore を **0** に設定すると、この制限が無効になります。デフォルトは **0** です。**podsPerCore** パラメーターの値は、**maxPods** パラメーターの値を超えることはできません。

maxPods パラメーターは、ノードのプロパティーに関係なく、ノードが実行できる Pod の数を固定値に設定します。

■

```
kubeletConfig:
  maxPods: 250
```

8.4.1. 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。

- 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推奨します。
- 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、**KubeletConfig** CR を 1 つ作成します。
- 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。

注記

kubelet またはコンテナーのランタイム設定をカスタムマシン設定プールに適用する場合、**machineConfigSelector** のカスタムロールは、カスタムマシン設定プールの名前と一致する必要があります。

たとえば、次のカスタムマシン設定プールの名前は **infra** であるため、カスタムロールも **infra** にする必要があります。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: infra
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,infra]}
# ...
```

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. 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。

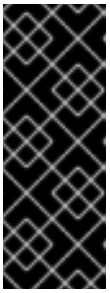
```
$ oc describe node ci-ln-5grqprb-f76d1-ncnqq-worker-a-mdv94
```

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。



重要

特定のマシン設定プールをターゲットとする kubelet 設定は、依存するプールにも影響します。たとえば、ワーカーノードを含むプール用の kubelet 設定を作成すると、インフラストラクチャーノードを含むプールを含むすべてのサブセットプールにも設定が適用されます。これを回避するには、ワーカーノードのみを含む選択式を使用して新しいマシン設定プールを作成し、kubelet 設定でこの新しいプールをターゲットにする必要があります。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
```

```
    custom-kubelet: set-max-pods (1)
kubeletConfig:
  maxPods: 500 (2)
```

1. Machine Config Pool からラベルを入力します。
2. kubelet 設定を追加します。この例では、**maxPods** を使用してノードあたりの最大 Pod を設定します。

注記

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

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

a. ラベルを使用してワーカーのマシン設定プールを更新します。

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

b. **KubeletConfig** オブジェクトを作成します。

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

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

```
$ oc get kubeletconfig
```

出力例

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

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

4. 変更がノードに適用されていることを確認します。

a. **maxPods** 値が変更されたワーカーノードで確認します。

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

b. **Allocatable** スタンザを見つけます。

```
 ...
Allocatable:
  attachable-volumes-gce-pd:  127
  cpu:                        3500m
  ephemeral-storage:          123201474766
  hugepages-1Gi:              0
  hugepages-2Mi:              0
  memory:                     14225400Ki
  pods:                       500 1
 ...
```

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

5. **KubeletConfig** オブジェクトの変更を確認します。

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

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
```

8.4.2. 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** フィールドを追加して、値を設定します。

```
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。

8.4.3. 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、ただし OVN-Kubernetes ネットワークプラグインを使用する場合は 24	64、ただし OVN-Kubernetes ネットワークプラグインを使用する場合は 128
501、ただし OVN-Kubernetes ネットワークプラグインではテストされていません	4000	16	96

上の表のデータは、r5.4xlarge インスタンスをコントロールプレーンノードとして使用し、m5.2xlarge インスタンスをワーカーノードとして使用する、AWS 上で実行される OpenShift Container Platform をベースとしています。

3 つのコントロールプレーン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、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、また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、コントロールプレーンモードが遮断 (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。

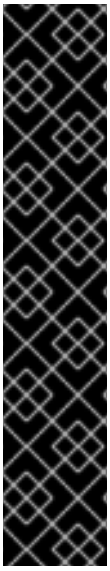
重要

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

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)
7000	6.6	12.9
8000	6.9	14.8
9000	8	17.7
10,000	9.9	21.6

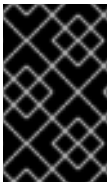


重要

以下の設定でのみ、実行中の OpenShift Container Platform 4.16 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。

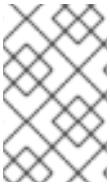
- ユーザーがプロビジョニングしたインストール方法でインストールされたクラスター。
- installer-provisioned infrastructure インストール方法でインストールされた AWS クラスター。
- コントロールプレーンマシンセットを使用してコントロールプレーンマシンを管理するクラスター。

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 3.11 以前のバージョンと比較すると、OpenShift Container Platform 4.16 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るようになりました。サイズはこれを考慮に入れて決定されます。

8.4.4. CPU マネージャーの設定

CPU マネージャーを設定するには、KubeletConfig カスタムリソース (CR) を作成し、それを目的のノードセットに適用します。

手順

1. 次のコマンドを実行してノードにラベルを付けます。

```
# oc label node perf-node.example.com cpumanager=true
```

2. すべてのコンピュートノードに対して CPU マネージャーを有効にするには、次のコマンドを実行して CR を編集します。

```
# oc edit machineconfigpool worker
```

3. **custom-kubelet: cpumanager-enabled** ラベルを **metadata.labels** セクションに追加します。

```
metadata:
  creationTimestamp: 2020-xx-xxx
  generation: 3
  labels:
    custom-kubelet: cpumanager-enabled
```

4. **KubeletConfig**、**cpumanager-kubeletconfig.yaml**、カスタムリソース (CR) を作成します。直前の手順で作成したラベルを参照し、適切なノードを新規の kubelet 設定で更新します。**machineConfigPoolSelector** セクションを参照してください。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
     cpuManagerPolicy: static 1
     cpuManagerReconcilePeriod: 5s 2
```

1. ポリシーを指定します。

   - **none**このポリシーは、既存のデフォルト CPU アフィニティースキームを明示的に有効にし、スケジューラーが自動的に実行するもの以外のアフィニティーを提供しません。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 内のコンテナーを許可します。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。

2. オプション: CPU マネージャーの調整頻度を指定します。デフォルトは **5s** です。

5. 次のコマンドを実行して、動的 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. 次のコマンドを実行して、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. 次のコマンドを実行して、更新された **kubelet.conf** ファイルをコンピュートノードで確認します。

```
# oc debug node/perf-node.example.com
sh-4.2# cat /host/etc/kubernetes/kubelet.conf | grep cpuManager
```

出力例

```
cpuManagerPolicy: static       1
cpuManagerReconcilePeriod: 5s  2
```

1. **cpuManagerPolicy** は、**KubeletConfig** CR の作成時に定義されます。
2. **cpuManagerReconcilePeriod** は、**KubeletConfig** CR の作成時に定義されます。

8. 次のコマンドを実行してプロジェクトを作成します。

```
$ oc new-project <project_name>
```

9. 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: cpumanager
    image: gcr.io/google_containers/pause:3.2
    resources:
      requests:
        cpu: 1
```

```
        memory: "1G"
      limits:
        cpu: 1
        memory: "1G"
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  nodeSelector:
    cpumanager: "true"
```

10. Pod を作成します。

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

検証

1. 次のコマンドを実行して、ラベルを付けたノードに Pod が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
```

2. 次のコマンドを実行して、CPU が Pod 専用として割り当てられていることを確認します。

```
# oc describe node --selector='cpumanager=true' | grep -i cpumanager- -B2
```

出力例

```
NAMESPACE    NAME                CPU Requests  CPU Limits  Memory Requests  Memory
Limits  Age
cpuman       cpumanager-mlrrz    1 (28%)       1 (28%)     1G (13%)         1G (13%)       27m
```

3. **cgroups** が正しく設定されていることを確認します。次のコマンドを実行して、**pause** プロセスのプロセス ID (PID) を取得します。

```
# oc debug node/perf-node.example.com
```

```
sh-4.2# systemctl status | grep -B5 pause
```

注記

出力で複数の pause プロセスエントリーが返される場合は、正しい一時停止プロセスを特定する必要があります。

出力例

```
# ├─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
```

4. 次のコマンドを実行して、サービス品質 (QoS) 層 (**Guaranteed**) の Pod が **kubepods.slice** サブディレクトリー内に配置されていることを確認します。

```
# cd /sys/fs/cgroup/kubepods.slice/kubepods-
pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice/crio-
b5437308f1ad1a7db0574c542bdf08563b865c0345c86e9585f8c0b0a655612c.scope
```

```
# for i in `ls cpuset.cpus cgroup.procs` ; do echo -n "$i "; cat $i ; done
```

注記

他の QoS 階層の Pod は、親 **kubepods** の子である **cgroups** に配置されます。

出力例

```
cpuset.cpus 1
tasks 32706
```

5. 次のコマンドを実行して、タスクに許可されている CPU リストを確認します。

```
# grep ^Cpus_allowed_list /proc/32706/status
```

出力例

```
 Cpus_allowed_list:    1
```

6. システム上の別の 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。たとえば、**besteffort** QoS 階層の Pod を検証するには、次のコマンドを実行します。

```
# cat /sys/fs/cgroup/kubepods.slice/kubepods-besteffort.slice/kubepods-besteffort-
podc494a073_6b77_11e9_98c0_06bba5c387ea.slice/crio-
c56982f57b75a2420947f0afc6cafe7534c5734efc34157525fa9abbf99e3849.scope/cpuset.cpus
```

```
# 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
```

8.5. HUGE PAGE

Huge Page について理解し、これを設定します。

8.5.1. 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ー (TLB: Translation Lookaside Buffer) は、仮想から物理へのページマッピン

グの小規模なハードウェアキャッシュのことです。ハードウェアの指示で渡された仮想アドレスが TLB にあれば、マッピングをすばやく決定できます。そうでない場合には、TLB ミスが発生し、システムは速度が遅く、ソフトウェアベースのアドレス変換にフォールバックされ、パフォーマンスの問題が発生します。TLB のサイズは固定されているので、TLB ミスの発生率を減らすには Page サイズを大きくする必要があります。

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

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

```
apiVersion: v1
kind: Pod
metadata:
  generateName: hugepages-volume-
spec:
  containers:
  - securityContext:
      privileged: true
    image: rhel7:latest
    command:
    - sleep
    - inf
    name: example
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        hugepages-2Mi: 100Mi (1)
        memory: "1Gi"
        cpu: "1"
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

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

指定されたサイズの Huge Page の割り当て

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

Huge page の要件

- Huge Page 要求は制限と同じでなければなりません。制限が指定されているにもかかわらず、要求が指定されていない場合には、これがデフォルトになります。
- Huge Page は、Pod のスコープで分割されます。コンテナーの分割は、今後のバージョンで予定されています。
- Huge Page がサポートする **EmptyDir** ボリュームは、Pod 要求よりも多くの Huge Page メモリーを消費することはできません。
- **shmget()** で **SHM_HUGETLB** を使用して Huge Page を消費するアプリケーションは、**proc/sys/vm/hugetlb_shm_group** に一致する補助グループで実行する必要があります。

8.5.3. 起動時の Huge Page 設定

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

手順

ノードの再起動を最小限にするには、以下の手順の順序に従う必要があります。

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

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

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

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: hugepages 1
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile: 2
  - data: |
```

```
    [main]
    summary=Boot time configuration for hugepages
    include=openshift-node
    [bootloader]
    cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50 3
  name: openshift-node-hugepages

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

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

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

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

4 マシン設定プールベースのマッチングを有効にします。

3. チューニングされた **hugepages** 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。

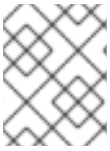
```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker-hp
  labels:
    worker-hp: ""
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,worker-hp]}
  nodeSelector:
    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ーが十分にある場合、**worker-hp** マシン設定プールのすべてのノードには 50 2Mi の Huge Page が割り当てられているはずです。

```
$ oc get node <node_using_hugepages> -o jsonpath="{.status.allocatable.hugepages-2Mi}"
100Mi
```



注記

TuneD ブートローダープラグインは、Red Hat Enterprise Linux CoreOS (RHCOS) ワーカーノードのみサポートします。

8.6. 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ジャーコール (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。

8.6.1. デバイスプラグインのデプロイ方法

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

8.6.2. Device Manager について

Device Manager 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ーは個別のベンダーによりサポートされます。

Device Manager 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 Device Manager 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 Device Manager に自己登録し、Device Manager 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。

Device Manager は、新規登録要求の処理時にデバイスプラグインサービスで **ListAndWatch** リモートプロシージャーコール (RPC) を起動します。応答として Device Manager 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。Device Manager 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 Device Manager 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 Device Manager に送信します。Device Manager 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。

8.6.3. Device Manager の有効化

Device Manager 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。

Device Manager 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 1
   ```

   1 Device Manager に必要なラベル。

手順

1. 設定変更のためのカスタムリソース (CR) を作成します。

   Device Manager CR の設定例

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

   1 CR に名前を割り当てます。

   2 Machine Config Pool からラベルを入力します。

3 **DevicePlugins** を 'true` に設定します。

2. Device Manager を作成します。

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

出力例

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

3. Device Manager 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、Device Manager 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、Device Manager 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。

8.7. 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。

8.7.1. 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 で構成されます。

表8.1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。 **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が、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	**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
#...
```

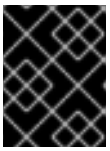
容認はテイントと一致します。

- **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。

8.7.2. 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" (1)
    value: "value1"
    operator: "Equal"
    effect: "NoExecute"
    tolerationSeconds: 3600 (2)
#...
```

(1) **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。

(2) **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" (1)
```

```
      effect: "NoExecute"
      tolerationSeconds: 3600
#...
```

1 **Exists** Operator は **value** を取りません。

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

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

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

以下に例を示します。

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

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

注記

NoSchedule テイントを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
#...
```

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

8.7.3. コンピュートマシンセットを使用したテイントおよび容認の追加

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

手順

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

   Equal 演算子を含む Pod 設定ファイルのサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1" 1
    value: "value1"
    operator: "Equal"
    effect: "NoExecute"
    tolerationSeconds: 3600 2
#...
```

   1 **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。

   2 **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。

   コンピュート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. コンピュートマシンセットを 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 を適用してコンピュート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. コンピュート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。

8.7.4. 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。

8.7.5. 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
#...
```

8.7.6. 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
#...
```

8.8. TOPOLOGY MANAGER

Topology Manager について理解し、これを使用します。

8.8.1. Topology Manager ポリシー

Topology Manager は、CPU マネージャーや Device Manager 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 Quality of Service (QoS) 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ーの場合、kubelet は各 Hint Provider を呼び出してそれらのリソースの可用性を検出します。この情報を使用して、Topology Manager は、そのコンテナーの推奨される NUMA ノードのアフィニティーを保存します。アフィニティーが優先されない場合、Topology Manager はこれを保管し、ノードに対して Pod を許可します。

restricted ポリシー

restricted トポロジー管理ポリシーを持つ Pod のそれぞれのコンテナーの場合、kubelet は各 Hint Provider を呼び出してそれらのリソースの可用性を検出します。この情報を使用して、Topology Manager は、そのコンテナーの推奨される NUMA ノードのアフィニティーを保存します。アフィニティーが優先されない場合、Topology Manager はこの Pod をノードから拒否します。これにより、Pod が Pod の受付の失敗により **Terminated** 状態になります。

single-numa-node ポリシー

single-numa-node トポロジー管理ポリシーがある Pod のそれぞれのコンテナーの場合、kubelet は各 Hint Provider を呼び出してそれらのリソースの可用性を検出します。この情報を使用して、Topology Manager は単一の NUMA ノードのアフィニティーが可能かどうかを判別します。可能である場合、Pod はノードに許可されます。単一の NUMA ノードアフィニティーが使用できない場合には、Topology Manager は Pod をノードから拒否します。これにより、Pod は Pod の受付失敗と共に Terminated (終了) 状態になります。

8.8.2. 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 1
    cpuManagerReconcilePeriod: 5s
    topologyManagerPolicy: single-numa-node 2
```

1. 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。
2. 選択した Topology Manager 割り当てポリシーを指定します。このポリシーは **single-numa-node** になります。使用できる値は、**default**、**best-effort**、**restricted**、**single-numa-node** です。

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

以下のサンプル **Pod** 仕様は、Pod の Topology Manger との対話について説明しています。

以下の Pod は、リソース要求や制限が指定されていないために **BestEffort** QoS クラスで実行されます。

```
spec:
  containers:
  - name: nginx
    image: nginx
```

以下の Pod は、要求が制限よりも小さいために **Burstable** QoS クラスで実行されます。

```
spec:
  containers:
```

```
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
```

選択したポリシーが **none** 以外の場合は、Topology Manager はこれらの **Pod** 仕様のいずれかも考慮しません。

以下の最後のサンプル Pod は、要求が制限と等しいために Guaranteed QoS クラスで実行されます。

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
      requests:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
```

Topology Manager はこの Pod を考慮します。Topology Manager はヒントプロバイダー (CPU マネージャーおよび Device Manager) 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、このコンテナーに最適なトポロジーを保管します。この Pod の場合、CPU マネージャーおよび Device Manager 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。

8.9. リソース要求とオーバーコミット

各コンピュートリソースについて、コンテナー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。コンテナー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。コンテナーは、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、コンピュートリソースのタイプによって異なります。コンテナーが要求または制限を指定しない場合、コンテナー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、コンテナーはローカルの最も低い優先順位で利用できる指定リソースを消費できます。リソースが不足する状態では、リソース要求を指定しないコンテナーに最低レベルの 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、コンテナーに 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、コンテナー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。

8.10. 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、コンテナー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)
      limitCPUToMemoryPercent: 200 (4)
# ...
```

(1) 名前は **cluster** でなければなりません。

(2) オプション: コンテナー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: コンテナーの 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: コンテナー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、制限がコンテナー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。

8.10.1. 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

- 制限がコンテナー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 のリストから **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 (1)
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 (2)
      cpuRequestToLimitPercent: 25 (3)
      limitCPUToMemoryPercent: 200 (4)
# ...
```

(1) 名前は **cluster** でなければなりません。

(2) オプション: コンテナー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: コンテナー 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: コンテナー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:
    kubectl.kubernetes.io/last-applied-configuration: |

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

```
      limitCPUToMemoryPercent: 200
      memoryRequestToLimitPercent: 50
status:

# ...

   mutatingWebhookConfigurationRef: 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。

8.10.2. 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

- 制限がコンテナー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。

1. Cluster Resource Override の namespace を作成します。
   a. 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
```

   b. 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
```

2. Operator グループを作成します。
   a. 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: "4.16"
  name: clusterresourceoverride
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. サブスクリプションを作成します。

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

以下に例を示します。

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

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

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

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

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

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
```

```
metadata:
  name: cluster (1)
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 (2)
      cpuRequestToLimitPercent: 25 (3)
      limitCPUToMemoryPercent: 200 (4)
```

(1) 名前は **cluster** でなければなりません。

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

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

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

以下に例を示します。

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

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

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

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

出力例

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

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","metadata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLimitPercent":50}}}}
  creationTimestamp: "2019-12-18T22:35:02Z"
  generation: 1
  name: cluster
```

```
  resourceVersion: "127622"
  selfLink: /apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
  uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
spec:
  podResourceOverride:
    spec:
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200
      memoryRequestToLimitPercent: 50
status:

# ...

    mutatingWebhookConfigurationRef: 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。

8.10.3. 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

- 制限がコンテナー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 1
      cpuRequestToLimitPercent: 25 2
      limitCPUToMemoryPercent: 200 3
# ...
```

1. オプション: コンテナー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。
2. オプション: コンテナー 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。
3. オプション: コンテナー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。

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のある各プロジェクトの namespace オブジェクトに追加されていることを確認します。

```
apiVersion: v1
kind: Namespace
metadata:

# ...

  labels:
    clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true" (1)

# ...
```

1. このラベルを各プロジェクトに追加します。

8.11. ノードレベルのオーバーコミット

quality of service (QOS) 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。

8.11.1. コンピュートリソースとコンテナーについて

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。

8.11.1.1. コンテナーの CPU 要求について

コンテナーには要求する CPU の量が保証され、さらにコンテナー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。複数のコンテナーが追加の CPU の使用を試行する場合、CPU 時間が各コンテナーで要求される CPU の量に基づいて分配されます。

たとえば、あるコンテナーが 500m の CPU 時間を要求し、別のコンテナーが 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 の比率でコンテナー間で分配されます。コンテナー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。

8.11.1.2. コンテナーのメモリー要求について

コンテナーには要求するメモリー量が保証されます。コンテナー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。コンテナー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。コンテナー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。

8.11.2. オーバーコミットメントと quality of service クラスについて

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

オーバーコミットされる環境では、ノード上の Pod がいずれかの時点で利用可能なコンピュートリソースよりも多くの量の使用を試行することができます。これが生じると、ノードはそれぞれの Pod に優先順位を指定する必要があります。この決定を行うために使用される機能は、Quality of Service (QoS) 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8.2 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ーが最初に強制終了されます。

- **Guaranteed** コンテナーは優先順位が最も高いコンテナーとして見なされ、保証されます。 強制終了されるのは、これらのコンテナーで制限を超えるか、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ーが他にない場合のみです。
- システム不足の状態にある **Burstable** コンテナーは、制限を超過し、**BestEffort** コンテナー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** コンテナーは優先順位の最も低いコンテナーとして処理されます。これらのコンテナーのプロセスは、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。

8.11.2.1. quality of service 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 OoS クラスの Pod が高い QoS クラスの Pod で要求されるリソースを使用できないようにするために要求されたリソースの予約を試行します。

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

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

- **qos-reserved=memory=50%** の値は、**Burstable** および **BestEffort** QoS クラスがこれらより高い QoS クラスによって要求されるメモリーの半分を消費することを許可します。

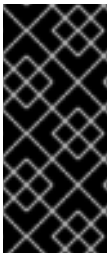
- **qos-reserved=memory=0%** の値は、**Burstable** および **BestEffort** QoS クラスがノードの割り当て可能分を完全に消費することを許可しますが (利用可能な場合)、これにより、**Guaranteed** ワークロードが要求したメモリーにアクセスできなくなるリスクが高まります。この状況により、この機能は無効にされています。

8.11.3. swap メモリーと QOS について

Quality of Service (QOS) 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ーが 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。

8.11.4. 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
```

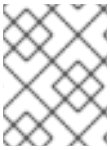
出力例

```
#...
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 層でのメモリー予約

8.11.5. 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。

- コンテナーに 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。
- コンテナーに 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。
- コンテナーに 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: "" (1)
  name: worker
```

(1) 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 (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" (2)
  kubeletConfig:
    cpuCfsQuota: false (3)
```

(1) CR に名前を割り当てます。

(2) マシン設定プールからラベルを指定します。

(3) **cpuCfsQuota** パラメーターを **false** 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
```

8.11.6. システムリソースのリソース予約

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

手順

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

8.11.7. 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
```

8.12. 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。

8.12.1. 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 オブジェクトファイルを作成または編集します。
2. 以下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。

8.13. 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。

8.13.1. 終了したコンテナー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

コンテナー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、終了したコンテナーを削除します。

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 のコンテナーが API から常にアクセス可能な状態になるよう試みます。Pod が削除された場合、コンテナーも削除されます。コンテナーは 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、コンテナーが削除され、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リストです。

表8.3 コンテナーのガ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。

8.13.2. イメージ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

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。

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8.4 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 つのイメージリスト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。

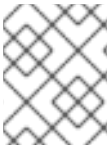
1. 1 つ以上の Pod で現在実行されているイメージのリスト
2. ホストで利用可能なイメージのリスト

新規コンテナー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。

8.13.3. コンテナー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。

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

コンテナーのガベージコレクションは終了したコンテナーを削除します。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: "" 1
  name: worker
#...
```

1 Labels の下にラベルが表示されます。

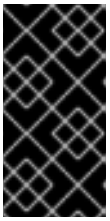
ヒント

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。

コンテナー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 (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" (2)
  kubeletConfig:
    evictionSoft: (3)
      memory.available: "500Mi" (4)
      nodefs.available: "10%"
      nodefs.inodesFree: "5%"
      imagefs.available: "15%"
      imagefs.inodesFree: "10%"
    evictionSoftGracePeriod: (5)
      memory.available: "1m30s"
      nodefs.available: "1m30s"
      nodefs.inodesFree: "1m30s"
      imagefs.available: "1m30s"
      imagefs.inodesFree: "1m30s"
    evictionHard: (6)
      memory.available: "200Mi"
      nodefs.available: "5%"
      nodefs.inodesFree: "4%"
      imagefs.available: "10%"
      imagefs.inodesFree: "5%"
    evictionPressureTransitionPeriod: 0s (7)
    imageMinimumGCAge: 5m (8)
    imageGCHighThresholdPercent: 80 (9)
    imageGCLowThresholdPercent: 75 (10)
#...
```

(1) オブジェクトの名前。

(2) マシン設定プールからラベルを指定します。

(3) コンテナー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**。

(4) コンテナー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。

(5) コンテナー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。

(6) コンテナー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。

(7) コンテナー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。

8 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。

9 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
```

8.14. NODE TUNING OPERATOR の使用

Node Tuning Operator について理解し、これを使用します。

目的
Node Tuning Operator は、TuneD デーモンを調整することでノードレベルのチューニングを管理し、パフォーマンスプロファイルコントローラーを使用して低レイテンシーのパフォーマンスを実現するのに役立ちます。ほとんどの高パフォーマンスアプリケーションでは、一定レベルのカーネルのチューニングが必要です。Node Tuning Operator は、ノードレベルの sysctl の統一された管理インターフェイスをユーザーに提供し、ユーザーが指定するカスタムチューニングを追加できるよう柔軟性を提供します。

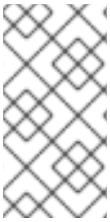
Operator は、コンテナー化された OpenShift Container Platform の TuneD デーモンを Kubernetes デーモンセットとして管理します。これにより、カスタムチューニング仕様が、デーモンが認識する形式でクラスターで実行されるすべてのコンテナー化された TuneD デーモンに渡されます。デーモンは、ノードごとに 1 つずつ、クラスターのすべてのノードで実行されます。

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

Node Tuning Operator は、パフォーマンスプロファイルコントローラーを使用して自動チューニングを実装し、OpenShift Container Platform アプリケーションの低レイテンシーパフォーマンスを実現します。

クラスター管理者は、以下のようなノードレベルの設定を定義するパフォーマンスプロファイルを設定します。

- カーネルを kernel-rt に更新します。
- ハウスキーピング用の CPU を選択します。
- 実行中のワークロード用の CPU を選択します。



注記

現在、CPU 負荷分散の無効化は cgroup v2 ではサポートされていません。その結果、cgroup v2 が有効になっている場合は、パフォーマンスプロファイルから望ましい動作が得られない可能性があります。パフォーマンスプロファイルを使用している場合は、cgroup v2 を有効にすることは推奨されません。

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。

注記

OpenShift Container Platform の以前のバージョンでは、Performance Addon Operator を使用して自動チューニングを実装し、OpenShift アプリケーションの低レイテンシーパフォーマンスを実現していました。OpenShift Container Platform 4.11 以降では、この機能は Node Tuning Operator の一部です。

8.14.1. 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

- 次のコマンドを実行して、Node Tuning Operator 仕様の例にアクセスします。

```
oc get tuned.tuned.openshift.io/default -o yaml -n openshift-cluster-node-tuning-operator
```

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

警告

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

8.14.2. 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リストです。2 つ目の **recommend:** は、プロファイル選択ロジックを定義します。

複数のカスタムチューニング仕様は、Operator の namespace に複数の CR として共存できます。新規 CR の存在または古い CR の削除は Operator によって検出されます。既存のカスタムチューニング仕様はすべてマージされ、コンテナー化された TuneD デーモンの適切なオブジェクトは更新されます。

管理状態

Operator 管理の状態は、デフォルトの Tuned CR を調整して設定されます。デフォルトで、Operator は Managed 状態であり、**spec.managementState** フィールドはデフォルトの Tuned CR に表示されません。Operator Management 状態の有効な値は以下のとおりです。

- Managed: Operator は設定リソースが更新されるとそのオペランドを更新します。
- Unmanaged: Operator は設定リソースへの変更を無視します。
- Removed: Operator は Operator がプロビジョニングしたオペランドおよびリソースを削除します。

プロファイルデータ

profile: セクションは、TuneD プロファイルおよびそれらの名前をリスト表示します。

```
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リストです。

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

リストの個別項目:

```
- machineConfigLabels: (1)
    <mcLabels> (2)
  match: (3)
    <match> (4)
  priority: <priority> (5)
  profile: <tuned_profile_name> (6)
  operand: (7)
    debug: <bool> (8)
    tunedConfig:
      reapply_sysctl: <bool> (9)
```

(1) オプション:

(2) キー/値の **MachineConfig** ラベルのディクショナリー。キーは一意である必要があります。

(3) 省略する場合は、優先度の高いプロファイルが最初に一致するか、**machineConfigLabels** が設定されていない限り、プロファイルの一致が想定されます。

(4) オプションのリスト。

(5) プロファイルの順序付けの優先度。数値が小さいほど優先度が高くなります (**0** が最も高い優先度になります)。

(6) 一致に適用する TuneD プロファイル。例: **tuned_profile_1**

(7) オプションのオペランド設定。

(8) TuneD デーモンのデバッグオンまたはオフを有効にします。オプションは、オンの場合は **true**、オフの場合は **false** です。デフォルトは **false** です。

(9) TuneD デーモンの **reapply_sysctl** 機能をオンまたはオフにします。オプションは on で **true**、オフの場合は **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>** リスト。

<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、リストは論理 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。

リスト項目の **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: ノードまたは Pod のラベルベースのマッチング

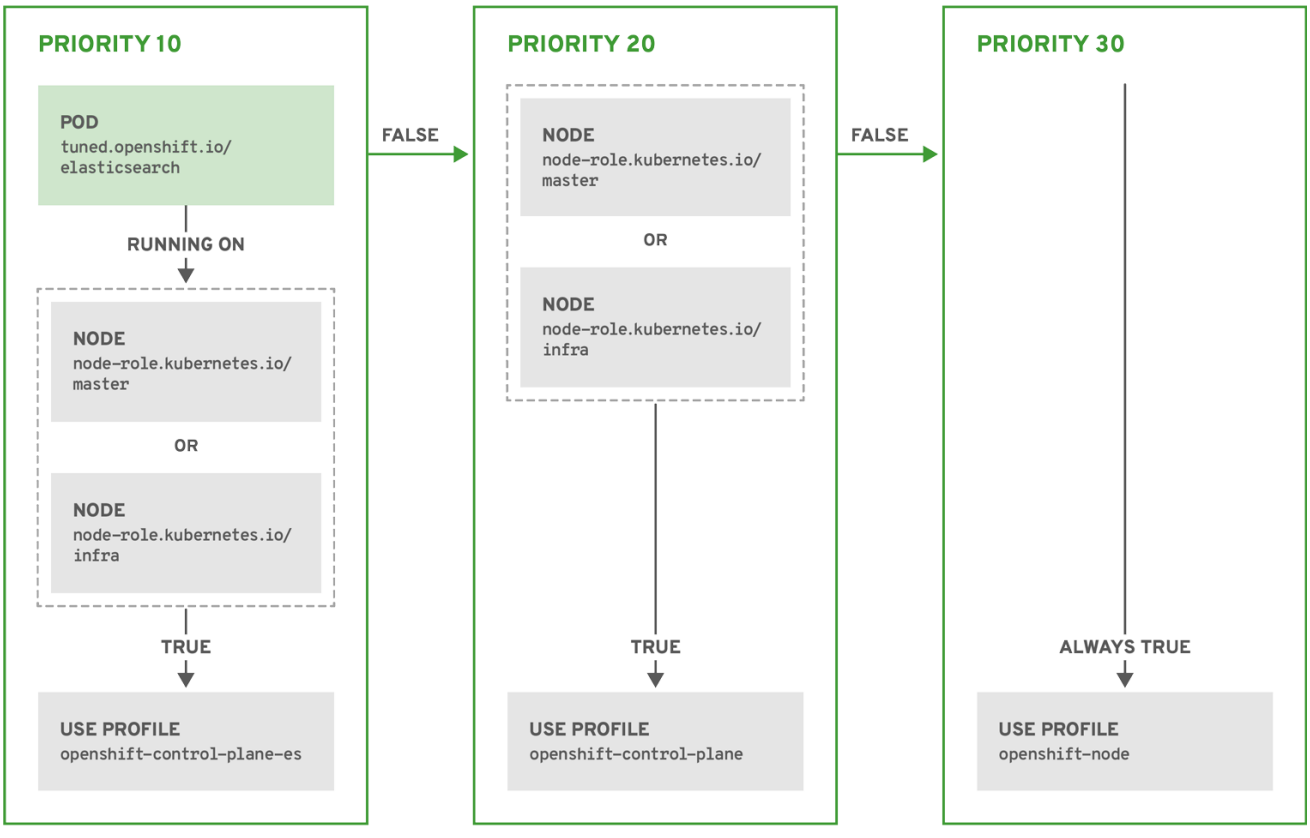
```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra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

```
    profile: openshift-control-plane
  - priority: 30
    profile: openshift-node
```

上記のコンテナー化された TuneD デーモンの CR は、プロファイルの優先順位に基づいてその **recommend.conf** ファイルに変換されます。最も高い優先順位 (**10**) を持つプロファイルは **openshift-control-plane-es** であるため、これが最初に考慮されます。指定されたノードで実行されるコンテナー化された TuneD デーモンは、同じノードに **tuned.openshift.io/elasticsearch** ラベルが設定された Pod が実行されているかどうかを確認します。これがない場合は、**<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、コンテナー化された 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。



例: マシン設定プールベースのマッチング

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
```

```
  name: openshift-node-custom
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=Custom OpenShift node profile with an additional kernel parameter
      include=openshift-node
      [bootloader]
      cmdline_openshift_node_custom=+skew_tick=1
    name: openshift-node-custom

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

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

クラウドプロバイダー固有の TuneD プロファイル

この機能により、すべてのクラウドプロバイダー固有のノードに、OpenShift Container Platform クラスター上の特定のクラウドプロバイダーに合わせて特別に調整された TuneD プロファイルを簡単に割り当てることができます。これは、追加のノードラベルを追加したり、ノードをマシン設定プールにグループ化したりせずに実行できます。

この機能は、**<cloud-provider>://<cloud-provider-specific-id>** の形式で **spec.providerID** ノードオブジェクト値を利用して、NTO オペランドコンテナーの **<cloud-provider>** の値で **/var/lib/tuned/provider** ファイルを書き込みます。その後、このファイルのコンテンツは TuneD により、プロバイダー **provider-<cloud-provider>** プロファイル (存在する場合) を読み込むために使用されます。

openshift-control-plane および **openshift-node** プロファイルの両方の設定を継承する **openshift** プロファイルは、条件付きプロファイルの読み込みを使用してこの機能を使用するよう更新されるようになりました。現時点で、NTO や TuneD にクラウドプロバイダー固有のプロファイルは含まれていません。ただし、すべてのクラウドプロバイダー固有のクラスターノードに適用されるカスタムプロファイル **provider-<cloud-provider>** を作成できます。

GCE クラウドプロバイダープロファイルの例

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: provider-gce
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=GCE Cloud provider-specific profile
      # Your tuning for GCE Cloud provider goes here.
    name: provider-gce
```

注記

プロファイルの継承により、**provider-<cloud-provider>** プロファイルで指定された設定は、**openshift** プロファイルとその子プロファイルによって上書きされます。

8.14.3. クラスターに設定されるデフォルトのプロファイル

以下は、クラスターに設定されるデフォルトのプロファイルです。

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=Optimize systems running OpenShift (provider specific parent profile)
      include=-provider-${f:exec:cat:/var/lib/ocp-tuned/provider},openshift
    name: openshift
  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 ^ {} \;
```

8.14.4. サポートされている TuneD デーモンプラグイン

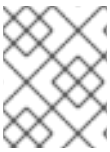
[main] セクションを除き、以下の TuneD プラグインは、Tuned CR の **profile:** セクションで定義されたカスタムプロファイルを使用する場合にサポートされます。

- 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



注記

TuneD ブートローダープラグインは、Red Hat Enterprise Linux CoreOS (RHCOS) ワーカーノードのみサポートします。

関連情報

- 利用可能な TuneD プラグイン
- TuneD を使い始める

8.15. 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 つのプロセッサー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: "" (1)
  name: worker
#...
```

(1) 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 (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" (2)
  kubeletConfig:
    podsPerCore: 10 (3)
    maxPods: 250 (4)
#...
```

(1) CR に名前を割り当てます。

(2) マシン設定プールからラベルを指定します。

(3) ノードがプロセッサーコアの数に基づいて実行できる Pod の数を指定します。

(4) ノードのプロパティーにかかわらず、ノードが実行できる Pod 数を固定値に指定します。

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
```

8.16. 静的 IP アドレスを使用したマシンのスケーリング

静的 IP アドレスを持つノードを実行するようにクラスターをデプロイした後、これらの静的 IP アドレスのいずれかを使用するようにマシンまたはマシンセットのインスタンスをスケーリングできます。

関連情報

- vSphere ノードの静的 IP アドレス

8.16.1. 静的 IP アドレスを使用するようにマシンをスケーリングする

追加のマシンセットを拡張して、クラスター上で事前定義された静的 IP アドレスを使用できます。この設定では、マシンリソース YAML ファイルを作成し、このファイルに静的 IP アドレスを定義する必要があります。

前提条件

- 設定された静的 IP アドレスを持つ少なくとも 1 つのノードを実行するクラスターをデプロイしました。

手順

1. マシンリソースの YAML ファイルを作成し、**network** パラメーターに静的 IP アドレスのネットワーク情報を定義します。

network パラメーターで定義された静的 IP アドレス情報を含むマシンリソース YAML ファイルの例。

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  creationTimestamp: null
  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>
  name: <infrastructure_id>-<role>
  namespace: openshift-machine-api
spec:
  lifecycleHooks: {}
  metadata: {}
  providerSpec:
    value:
      apiVersion: machine.openshift.io/v1beta1
      credentialsSecret:
        name: vsphere-cloud-credentials
      diskGiB: 120
      kind: VSphereMachineProviderSpec
      memoryMiB: 8192
      metadata:
        creationTimestamp: null
      network:
        devices:
        - gateway: 192.168.204.1 (1)
          ipAddrs:
          - 192.168.204.8/24 (2)
          nameservers: (3)
          - 192.168.204.1
          networkName: qe-segment-204
      numCPUs: 4
      numCoresPerSocket: 2
      snapshot: ""
      template: <vm_template_name>
      userDataSecret:
        name: worker-user-data
      workspace:
        datacenter: <vcenter_datacenter_name>
        datastore: <vcenter_datastore_name>
        folder: <vcenter_vm_folder_path>
        resourcepool: <vsphere_resource_pool>
        server: <vcenter_server_ip>
status: {}
```

(1) ネットワークインターフェイスのデフォルトゲートウェイの IP アドレス。

2 インストールプログラムがネットワークインターフェイスに渡す IPv4、IPv6、またはその両方の IP アドレスをリストします。どちらの IP ファミリーも、デフォルトネットワーク

3 DNS ネームサーバーをリストします。最大 3 つの DNS ネームサーバーを定義できます。1 つの DNS ネームサーバーが到達不能になった場合に、DNS 解決を利用できるように、複数の DNS ネームサーバーを定義することを検討してください。

- ターミナルに次のコマンドを入力して、**machine** のカスタムリソース (CR) を作成します。

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

8.16.2. 静的 IP アドレスが設定されたマシンのマシンセットスケーリング

マシンセットを使用して、設定された静的 IP アドレスを持つマシンをスケールすることができます。

マシンの静的 IP アドレスを要求するようにマシンセットを設定した後、マシンコントローラーは **openshift-machine-api** namespace に **IPAddressClaim** リソースを作成します。次に、外部コントローラーは **IPAddress** リソースを作成し、静的 IP アドレスを **IPAddressClaim** リソースにバインドします。

重要

組織では、さまざまな種類の IP アドレス管理 (IPAM) サービスを使用している場合があります。OpenShift Container Platform で特定の IPAM サービスを有効にする場合は、YAML 定義で **IPAddressClaim** リソースを手動で作成し、**oc** CLI で次のコマンドを入力してこのリソースに静的 IP アドレスをバインドしないといけない場合があります。

```
$ oc create -f <ipaddressclaim_filename>
```

次に、**IPAddressClaim** リソースの例を示します。

```
kind: IPAddressClaim
metadata:
  finalizers:
  - machine.openshift.io/ip-claim-protection
  name: cluster-dev-9n5wg-worker-0-m7529-claim-0-0
  namespace: openshift-machine-api
spec:
  poolRef:
    apiGroup: ipamcontroller.example.io
    kind: IPPool
    name: static-ci-pool
status: {}
```

マシンコントローラーはマシンを **IPAddressClaimed** のステータスで更新し、静的 IP アドレスが **IPAddressClaim** リソースに正常にバインドされたことを示します。マシンコントローラーは、バインドされた静的 IP アドレスをそれぞれに含む複数の **IPAddressClaim** リソースを持つマシンに同じステータスを適用します。その後、マシンコントローラーは仮想マシンを作成し、マシンの設定の **providerSpec** にリストされているすべてのノードに静的 IP アドレスを適用します。

8.16.3. マシンセットを使用して設定された静的 IP アドレスを持つマシンをスケールする

マシンセットを使用して、設定された静的 IP アドレスを持つマシンをスケールすることができます。

この手順の例では、マシンセット内のマシンをスケーリングするためのコントローラーの使用方法を示します。

前提条件

- 設定された静的 IP アドレスを持つ少なくとも 1 つのノードを実行するクラスターをデプロイしました。

手順

1. マシンセットの YAML ファイルの **network.devices.addressesFromPools** スキーマに IP プール情報を指定して、マシンセットを設定します。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  annotations:
    machine.openshift.io/memoryMb: "8192"
    machine.openshift.io/vCPU: "4"
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id>
  name: <infrastructure_id>-<role>
  namespace: openshift-machine-api
spec:
  replicas: 0
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  template:
    metadata:
      labels:
        ipam: "true"
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
    spec:
      lifecycleHooks: {}
      metadata: {}
      providerSpec:
        value:
          apiVersion: machine.openshift.io/v1beta1
          credentialsSecret:
            name: vsphere-cloud-credentials
          diskGiB: 120
          kind: VSphereMachineProviderSpec
          memoryMiB: 8192
          metadata: {}
          network:
            devices:
```

```
      - addressesFromPools: (1)
        - group: ipamcontroller.example.io
          name: static-ci-pool
          resource: IPPool
        nameservers:
        - "192.168.204.1" (2)
        networkName: qe-segment-204
    numCPUs: 4
    numCoresPerSocket: 2
    snapshot: ""
    template: rvanderp4-dev-9n5wg-rhcos-generated-region-generated-zone
    userDataSecret:
      name: worker-user-data
    workspace:
      datacenter: IBMCdatacenter
      datastore: /IBMCdatacenter/datastore/vsanDatastore
      folder: /IBMCdatacenter/vm/rvanderp4-dev-9n5wg
      resourcePool: /IBMCdatacenter/host/IBMCcluster//Resources
      server: vcenter.ibmc.devcluster.openshift.com
```

(1) 静的 IP アドレスまたは静的 IP アドレスの範囲をリストする IP プールを指定します。IP プールは、カスタムリソース定義 (CRD) への参照、または **IPAddressClaims** リソースハンドラーによってサポートされるリソースのいずれかになります。マシンコントローラーは、マシンセットの設定にリストされている静的 IP アドレスにアクセスし、各アドレスを各マシンに割り当てます。

(2) ネームサーバーをリストします。Dynamic Host Configuration Protocol (DHCP) ネットワーク設定は静的 IP アドレスをサポートしていないため、静的 IP アドレスを受け取るノードにはネームサーバーを指定する必要があります。

2. **oc** CLI で次のコマンドを入力して、マシンセットをスケールします。

```
$ 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
```

各マシンがスケールアップされた後、マシンコントローラーは **IPAddresssClaim** リソースを作成します。

3. オプション: 次のコマンドを入力して、**IPAddressClaim** リソースが **openshift-machine-api** namespace に存在することを確認します。

```
$ oc get ipaddressclaims.ipam.cluster.x-k8s.io -n openshift-machine-api
```

openshift-machine-api namespace にリストされている 2 つの IP プールをリストする oc CLI 出力の例

```
NAME                                      POOL NAME       POOL KIND
cluster-dev-9n5wg-worker-0-m7529-claim-0-0   static-ci-pool   IPPool
cluster-dev-9n5wg-worker-0-wdqkt-claim-0-0   static-ci-pool   IPPool
```

4. 次のコマンドを入力して、**IPAddress** リソースを作成します。

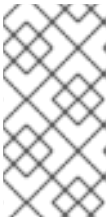
```
$ oc create -f ipaddress.yaml
```

次の例は、定義されたネットワーク設定情報と 1 つの静的 IP アドレスが定義された **IPAddress** リソースを示しています。

```
apiVersion: ipam.cluster.x-k8s.io/v1alpha1
kind: IPAddress
metadata:
  name: cluster-dev-9n5wg-worker-0-m7529-ipaddress-0-0
  namespace: openshift-machine-api
spec:
  address: 192.168.204.129
  claimRef: 1
    name: cluster-dev-9n5wg-worker-0-m7529-claim-0-0
  gateway: 192.168.204.1
  poolRef: 2
    apiGroup: ipamcontroller.example.io
    kind: IPPool
    name: static-ci-pool
  prefix: 23
```

1 ターゲットの **IPAddressClaim** リソースの名前。

2 ノードからの静的 IP アドレスに関する詳細情報。



注記

デフォルトでは、外部コントローラーはマシンセット内のリソースを自動的にスキャンして、認識可能なアドレスプールタイプを探します。外部コントローラーが **IPAddress** リソースで定義された **kind: IPPool** を見つけると、コントローラーは静的 IP アドレスを **IPAddressClaim** リソースにバインドします。

5. **IPAddress** リソースへの参照を使用して **IPAddressClaim** ステータスを更新します。

```
$ oc --type=merge patch IPAddressClaim cluster-dev-9n5wg-worker-0-m7529-claim-0-0 -
p='{"status":{"addressRef": {"name": "cluster-dev-9n5wg-worker-0-m7529-ipaddress-0-0"}}}' -
n openshift-machine-api --subresource=status
```

第9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。

9.1. 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ます。

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

クラスターネットワークプラグイン。**OVNKubernetes** は、インストール時にサポートされる唯一のプラグインです。

注記

クラスターをインストールした後は、**clusterNetwork** IP アドレス範囲のみ変更できます。デフォルトのネットワークタイプは、移行時に OpenShift SDN から OVN-Kubernetes にのみ変更できます。

9.2. クラスター全体のプロキシーの有効化

Proxy オブジェクトは、クラスター全体の egress プロキシーを管理するために使用されます。プロキシー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 のプロキシーを設定できます。

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、追加のプロキシー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 接続のプロキシーに必要な追加の CA 証明書が含まれる config map を作成します。

注記

プロキシーのアイデンティティー証明書が 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) プロキシーのアイデンティティー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 config map 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. プロキシー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ー 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ー 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ーの URL を指定します。たとえば、ほとんどのプロキシーは、**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ーを使用している場合は、プロキシーが使用する 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. プロキシー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ーをバイパスします。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、それらをこのリストに追加し、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 接続のプロキシーに必要な追加の 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 が存在している必要があります。このフィールドは、プロキシーのアイデンティティー証明書が 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。

9.3. 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
```

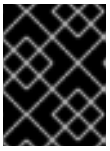
```
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructure_id>-int
      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>-wfpg4: owned
status: {}
```

9.4. 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の設定	クラスターのすべてのノードでサービスを公開します。

9.5. 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。

9.5.1. 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。

9.5.1.1. 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"]
```

9.6. IPSEC 暗号化の設定

IPsec を有効にすると、OVN-Kubernetes クラスターネットワークプラグイン上のノード間のすべてのネットワークトラフィックは、暗号化されたトンネルを通過します。

IPsec はデフォルトで無効にされています。

9.6.1. 前提条件

- クラスターは OVN-Kubernetes ネットワークプラグインを使用する必要がある。

9.6.1.1. IPsec 暗号化の有効化

クラスター管理者は、Pod 間の IPsec 暗号化、およびクラスターと外部 IPsec エンドポイント間の IPsec 暗号化を有効にすることができます。

次のいずれかのモードで IPsec を設定できます。

- **Full**: Pod 間のトラフィックおよび外部トラフィックの暗号化
- **External**: 外部トラフィックの暗号化

Pod 間のトラフィックに加えて外部トラフィックの暗号化を設定する必要がある場合は、「外部トラフィックの IPsec 暗号化の設定」の手順も完了する必要があります。

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している。
- クラスター MTU のサイズを **46** バイト減らして、IPsec ESP ヘッダーにオーバーヘッドを設けている。

手順

1. IPsec 暗号化を有効にするには、次のコマンドを入力します。

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
-p '{
  "spec":{
    "defaultNetwork":{
      "ovnKubernetesConfig":{
        "ipsecConfig":{
          "mode":<mode>
        }}}}}'
```

   ここでは、以下のようになります。

   mode

   外部ホストへのトラフィックのみを暗号化するには **External** を指定します。Pod 間のトラフィックを暗号化し、必要に応じて外部ホストへのトラフィックを暗号化するには **Full** を指定します。デフォルトでは、IPsec は無効になっています。

2. オプション: 外部ホストへのトラフィックを暗号化する必要がある場合は、「外部トラフィックの IPsec 暗号化の設定」の手順を実行します。

検証

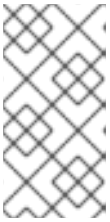
1. OVN-Kubernetes データプレーン Pod の名前を見つけるには、次のコマンドを入力します。

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

   出力例

```
ovnkube-node-5xqbf                 8/8     Running   0          28m
ovnkube-node-6mwcx                 8/8     Running   0          29m
ovnkube-node-ck5fr                 8/8     Running   0          31m
ovnkube-node-fr4ld                 8/8     Running   0          26m
ovnkube-node-wgs4l                 8/8     Running   0          33m
ovnkube-node-zfvcl                 8/8     Running   0          34m
```

2. 次のコマンドを実行して、クラスターで IPsec が有効になっていることを確認します。



注記

クラスター管理者は、IPsec が **Full** モードで設定されている場合に、クラスター上の Pod 間で IPsec が有効になっていることを確認できます。この手順では、クラスターと外部ホストの間で IPsec が機能しているかどうかは検証されません。

```
$ oc -n openshift-ovn-kubernetes rsh ovnkube-node-<XXXXX> ovn-nbctl --no-leader-only get nb_global . ipsec
```

ここでは、以下のようになります。

<XXXXX>
前の手順の Pod の文字のランダムなシーケンスを指定します。

出力例

```
true
```

9.7. 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。

9.7.1. 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ネットワークプラグインを使用するクラスターでは、ネットワーク分離は **NetworkPolicy** オブジェクトによって完全に制御されます。OpenShift Container Platform 4.16 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。

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、ICMP、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 Controller からの接続のみを許可します。
  プロジェクトで OpenShift Container Platform Ingress Controller 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て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。

9.7.1.1. allow-from-router ネットワークポリシーの使用

次の **NetworkPolicy** を使用して、ルーターの設定に関係なく外部トラフィックを許可します。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-router
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/ingress: "" 1
  podSelector: {}
  policyTypes:
  - Ingress
```

1 **policy-group.network.openshift.io/ingress:""** ラベルは、OpenShift-SDN と OVN-Kubernetes の両方をサポートします。

9.7.1.2. allow-from-hostnetwork ネットワークポリシーの使用

次の **allow-from-hostnetwork NetworkPolicy** オブジェクトを追加して、ホストネットワーク Pod からのトラフィックを転送します。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-hostnetwork
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/host-network: ""
  podSelector: {}
  policyTypes:
  - Ingress
```

9.7.2. サンプル NetworkPolicy オブジェクト

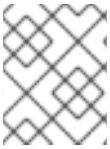
以下は、サンプル NetworkPolicy オブジェクトにアノテーションを付けます。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 (1)
spec:
  podSelector: (2)
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: (3)
        matchLabels:
          app: app
    ports: (4)
    - protocol: TCP
      port: 27017
```

1. NetworkPolicy オブジェクトの名前。
2. 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。
3. 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 namaspace にある Pod を照合して検索します。
4. 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。

9.7.3. CLI を使用した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ネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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。

   これは基本的なポリシーであり、他のネットワークポリシーの設定によって許可されたクロス Pod トラフィック以外のすべてのクロス Pod ネットワーキングをブロックします。

   ```
   kind: NetworkPolicy
   apiVersion: networking.k8s.io/v1
   metadata:
     name: deny-by-default
   spec:
     podSelector: {}
     policyTypes:
     - Ingress
     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: {}
   ```

   特定の namespace から 1つの Pod への上りトラフィックを許可する

   このポリシーは、**namespace-y** で実行されている Pod から **pod-a** というラベルの付いた Pod へのトラフィックを許可します。

   ```
   kind: NetworkPolicy
   apiVersion: networking.k8s.io/v1
   metadata:
     name: allow-traffic-pod
   spec:
     podSelector:
       matchLabels:
   ```

```
    pod: pod-a
policyTypes:
- Ingress
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        kubernetes.io/metadata.name: namespace-y
```

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>

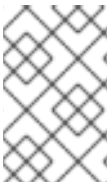
ネットワークポリシーファイル名を指定します。

<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/deny-by-default 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。

9.7.4. 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。

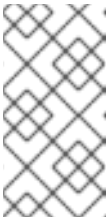
前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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
```

9.7.5. 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。

9.7.6. 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。

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. **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>
   # ...
   ```

7. 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。

9.7.6.1. 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、**mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプラグインなど、**NetworkPolicy** オブジェクトをサポートするデフォルトの CNI ネットワークプラグインを使用します。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

1. 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。

2. 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> 1
   ```

   1 **<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
   ```

9.8. 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。

9.8.1. ベースライン Ingress Controller (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

暗号化	LoadBalancerService	HostNetwork
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、8 kB という小さなページサイズで 1 Gbit の NIC を飽和させることができます。

最新のプロセッサーが搭載されたベアメタルで実行する場合は、上記のパブリッククラウドインスタンスのパフォーマンスの約 2 倍のパフォーマンスになることを予想できます。このオーバーヘッドは、パブリッククラウドにある仮想化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ー
100-1000	動的なコンテンツを生成するアプリケーション

通常、HAProxy は、使用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 Controller 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。

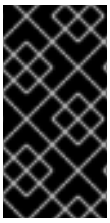
9.8.2. Ingress コントローラー (ルーター) liveness、readiness、および startup プローブの設定

クラスター管理者は、OpenShift Container Platform Ingress Controller (ルーター) によって管理されるルーター展開の kubelet の活性、準備、およびスタートアッププローブのタイムアウト値を設定できます。ルーターの liveness および readiness プローブは、デフォルトのタイムアウト値である 1 秒を使用します。これは、ネットワークまたはランタイムのパフォーマンスが著しく低下している場合には短すぎます。プローブのタイムアウトにより、アプリケーション接続を中断する不要なルーターの再起動が発生する可能性があります。より大きなタイムアウト値を設定する機能により、不要で不要な再起動のリスクを減らすことができます。

ルーターコンテナーの **livenessProbe**、**readinessProbe**、および **startupProbe** パラメーターの **timeoutSeconds** 値を更新できます。

パラメーター	説明
livenessProbe	**livenessProbe** は、Pod が停止していて再起動が必要かどうかを kubelet に報告します。

パラメーター	説明
readinessProbe	**readinessProbe** は、Pod が正常かどうかを報告します。準備プローブが異常な Pod を報告すると、kubelet は Pod をトラフィックを受け入れる準備ができていないものとしてマークします。その後、その Pod のエンドポイントは準備ができていないとマークされ、このステータスが kube-proxy に伝播されます。ロードバランサーが設定されたクラウドプラットフォームでは、kube-proxy はクラウドロードバランサーと通信して、その Pod を持つノードにトラフィックを送信しません。
startupProbe	**startupProbe** は、kubelet がルーターの活性と準備のプローブの送信を開始する前に、ルーター Pod の初期化に最大 2 分を与えます。この初期化時間により、多くのルートまたはエンドポイントを持つルーターが時期尚早に再起動するのを防ぐことができます。



重要

タイムアウト設定オプションは、問題を回避するために使用できる高度なチューニング手法です。ただし、これらの問題は最終的に診断する必要があり、プローブがタイムアウトする原因となる問題については、サポートケースまたは Jira issue を開く必要があります。

次の例は、デフォルトのルーター展開に直接パッチを適用して、活性プローブと準備プローブに 5 秒のタイムアウトを設定する方法を示しています。

```
$ oc -n openshift-ingress patch deploy/router-default --type=strategic --patch='{"spec":{"template":
{"spec":{"containers":[{"name":"router","livenessProbe":{"timeoutSeconds":5},"readinessProbe":
{"timeoutSeconds":5}}]}}}}'
```

検証

```
$ oc -n openshift-ingress describe deploy/router-default | grep -e Liveness: -e Readiness:
    Liveness:   http-get http://:1936/healthz delay=0s timeout=5s period=10s #success=1 #failure=3
    Readiness:  http-get http://:1936/healthz/ready delay=0s timeout=5s period=10s #success=1
#failure=3
```

9.8.3. HAProxy リロード間隔の設定

ルートまたはルートに関連付けられたエンドポイントを更新すると、OpenShift Container Platform ルーターは HAProxy の設定を更新します。次に、HAProxy は更新された設定をリロードして、これらの変更を有効にします。HAProxy がリロードすると、更新された設定を使用して新しい接続を処理する新しいプロセスが生成されます。

HAProxy は、それらの接続がすべて閉じられるまで、既存の接続を処理するために古いプロセスを実行し続けます。古いプロセスの接続が長く続くと、これらのプロセスはリソースを蓄積して消費する可能性があります。

デフォルトの最小 HAProxy リロード間隔は 5 秒です。**spec.tuningOptions.reloadInterval** フィールドを使用して Ingress Controller を設定し、より長い最小リロード間隔を設定できます。

警告

最小 HAProxy リロード間隔に大きな値を設定すると、ルートとそのエンドポイントの更新を監視する際にレイテンシーが発生する可能性があります。リスクを軽減するには、更新の許容レイテンシーよりも大きな値を設定しないようにしてください。

手順

- 次のコマンドを実行して、Ingress Controller のデフォルト最小 HAProxy リロード間隔を 15 秒に変更します。

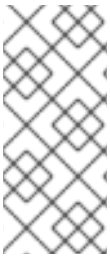
```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --patch='{"spec":{"tuningOptions":{"reloadInterval":"15s"}}}'
```

9.9. 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。

9.9.1. 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。

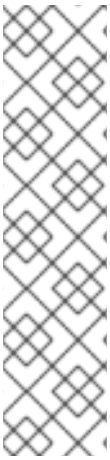
```
$ openstack port show <cluster_name>-<cluster_ID>-ingress-port
```

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> <app name>.apps.<cluster name>.<base domain>
```

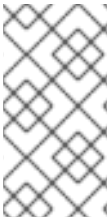
9.9.2. OVS ハードウェアオフロードの有効化

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。



注記

アプリケーション層のゲートウェイフローは、OpenShift Container Platform バージョン 4.10、4.11、および 4.12 では機能しません。また、OpenShift Container Platform バージョン 4.13 のアプリケーション層のゲートウェイフローをオフロードすることはできません。

手順

1. 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"
```

1 **SriovNetworkNodePolicy** の値をここに挿入します。

2 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 1
metadata:
  name: "hwoffload10"
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  isRdma: true
  nicSelector:
    pfNames: 2
    - ens5
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: 'true'
  numVfs: 1
  priority: 99
  resourceName: "hwoffload10"
```

1 **SriovNetworkNodePolicy** の値をここに挿入します。

2 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. 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"
    }'
```

3. 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/resourceName: openshift.io/hwoffload9
    k8s.v1.cni.cncf.io/resourceName: openshift.io/hwoffload10
spec:
  restartPolicy: Never
  containers:
  - name: dpdk-testpmd
    image: quay.io/krister/centos8_nfv-container-dpdk-testpmd:latest
```

9.9.3. OVS ハードウェアオフロードネットワークの接続

Open vSwitch (OVS) ハードウェアオフロードネットワークをクラスターに接続できます。

前提条件

- クラスターがインストールされ、実行されている。
- クラスターで使用するために、Red Hat OpenStack Platform (RHOSP) で OVS ハードウェアオフロードネットワークをプロビジョニングしている。

手順

1. 次のテンプレートから **network.yaml** という名前のファイルを作成します。

```
spec:
  additionalNetworks:
  - name: hwoffload1
    namespace: cnf
    rawCNIConfig: '{ "cniVersion": "0.3.1", "name": "hwoffload1", "type": "host-
device","pciBusId": "0000:00:05.0", "ipam": {}}' 1
    type: Raw
```

ここでは、以下のようになります。

pciBusId

オフロードネットワークに接続されているデバイスを指定します。この値がわからない場合は、次のコマンドを実行してこの値を見つけることができます。

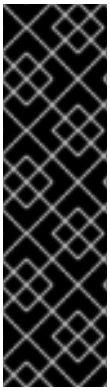
```
$ oc describe SriovNetworkNodeState -n openshift-sriov-network-operator
```

2. コマンドラインから次のコマンドを入力して、ファイルを使用してクラスターにパッチを適用します。

```
$ oc apply -f network.yaml
```

9.9.4. RHOSP で Pod への IPv6 接続を有効にする

異なるノード上にある追加のネットワークを持つ Pod 間の IPv6 接続を有効にするには、サーバーの IPv6 ポートのポートセキュリティーを無効にします。ポートセキュリティーを無効にすると、Pod に割り当てられた IPv6 アドレスごとに許可されたアドレスペアを作成する必要がなくなり、セキュリティーグループのトラフィックが有効になります。



重要

次の IPv6 追加ネットワーク設定のみがサポートされています。

- SLAAC とホストデバイス
- SLAAC と MACVLAN
- DHCP ステートレスおよびホストデバイス
- DHCP ステートレスおよび MACVLAN

手順

- コマンドラインで、次のコマンドを入力します。

```
$ openstack port set --no-security-group --disable-port-security <compute_ipv6_port>
```

重要

このコマンドは、ポートからセキュリティーグループを削除し、ポートセキュリティーを無効にします。トラフィックの制限は、ポートから完全に削除されます。

ここでは、以下のようになります。

<compute_ipv6_port>
コンピュートサーバーの IPv6 ポートを指定します。

9.9.5. RHOSP 上の Pod への IPv6 接続の追加

Pod で IPv6 接続を有効にしたら、Container Network Interface (CNI) 設定を使用して Pod に接続を追加します。

手順

1. Cluster Network Operator (CNO) を編集するには、次のコマンドを入力します。

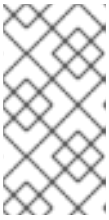
```
$ oc edit networks.operator.openshift.io cluster
```

2. **spec** フィールドで CNI 設定を指定します。たとえば、次の設定では、MACVLAN で SLAAC アドレスモードを使用します。

```
...
spec:
  additionalNetworks:
  - name: ipv6
    namespace: ipv6 1
    rawCNIConfig: '{ "cniVersion": "0.3.1", "name": "ipv6", "type": "macvlan", "master": "ens4"}'
2
    type: Raw
```

1 1 必ず同じ namespace に Pod を作成してください。

2 より多くのネットワークが設定されている場合、または別のカーネルドライバーが使用されている場合、ネットワークアタッチメントの **"master"** フィールドのインターフェイスは **"ens4"** とは異なる場合があります。



注記

ステートフルアドレスモードを使用している場合は、CNI 設定に IP アドレス管理 (IPAM) を含めます。

Multus は DHCPv6 をサポートしていません。

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。

検証

- コマンドラインで、次のコマンドを入力します。

```
$ oc get network-attachment-definitions -A
```

出力例

```
NAMESPACE      NAME          AGE
ipv6           ipv6          21h
```

セカンダリー IPv6 接続を持つ Pod を作成できるようになりました。

関連情報

- ネットワーク追加割り当ての設定

9.9.6. RHOSP で IPv6 接続を持つ Pod の作成

Pod の IPv6 接続を有効にして Pod に追加したら、セカンダリー IPv6 接続を持つ Pod を作成します。

手順

1. IPv6 namespace とアノテーション **k8s.v1.cni.cncf.io/networks: <additional_network_name>** を使用する Pod を定義します。ここで、**<additional_network_name** は追加のネットワークの名前になります。たとえば、**Deployment** オブジェクトの一環として、以下を行います。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-openshift
  namespace: ipv6
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values:
            - hello-openshift
  replicas: 2
  selector:
    matchLabels:
      app: hello-openshift
  template:
    metadata:
      labels:
        app: hello-openshift
      annotations:
        k8s.v1.cni.cncf.io/networks: ipv6
    spec:
      securityContext:
        runAsNonRoot: true
        seccompProfile:
          type: RuntimeDefault
      containers:
      - name: hello-openshift
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
```

```
      drop:
      - ALL
  image: quay.io/openshift/origin-hello-openshift
  ports:
  - containerPort: 8080
```

2. Pod を作成します。たとえば、コマンドラインで次のコマンドを入力します。

```
$ oc create -f <ipv6_enabled_resource>
```

ここでは、以下のようになります。

<ipv6_enabled_resource>
リソース定義を含むファイルを指定します。

第10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。

10.1. 動的プロビジョニング

10.1.1. 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。

10.1.2. 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。
Amazon Elastic Block Store (Amazon 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。

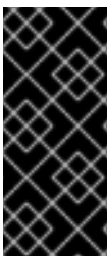
ストレージタイプ	プロビジョナープラグインの名前	注記
Azure Disk	**kubernetes.io/azure-disk**	
Azure File	**kubernetes.io/azure-file**	**persistent-volume-binder** サービスアカウントでは、Azure ストレージアカウントおよびキーを保存するためにシークレットを作成し、取得するためのパーミッションが必要です。
GCE Persistent Disk (gcePD)	**kubernetes.io/gce-pd**	マルチゾーン設定では、GCE プロジェクトごとに OpenShift Container 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。
IBM Power® 仮想サーバーブロック	**powervs.csi.ibm.com**	インストール後、IBM Power® Virtual Server Block CSI Driver Operator と IBM Power® Virtual Server Block CSI Driver は、動的プロビジョニングに必要なストレージクラスを自動的に作成します。
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もあります。

10.2. 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。

10.2.1. 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 (1)
apiVersion: storage.k8s.io/v1 (2)
metadata:
  name: <storage-class-name> (3)
  annotations: (4)
    storageclass.kubernetes.io/is-default-class: 'true'
    ...
provisioner: kubernetes.io/aws-ebs (5)
parameters: (6)
  type: gp3
...
```

(1) (必須) API オブジェクトタイプ。

(2) (必須) 現在の apiVersion。

(3) (必須) ストレージクラスの名前。

(4) (オプション) ストレージクラスのアノテーション。

(5) (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。

(6) (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。

10.2.2. ストレージクラスのアノテーション

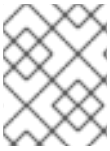
ストレージクラスをクラスター全体のデフォルトとして設定するには、以下のアノテーションをストレージクラスのメタデータに追加します。

```
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ク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
...
```

10.2.3. 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> 1
provisioner: kubernetes.io/cinder
parameters:
  type: fast 2
  availability: nova 3
  fsType: ext4 4
```

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. Cinder で作成されるボリュームタイプ。デフォルトは空です。
3. アベイラビリティーゾーン。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。
4. 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。

10.2.4. 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> 1
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 2
  iopsPerGB: "10" 3
  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**、**gp3**、**sc1**、**st1** から選択します。デフォルトは **gp3** です。有効な Amazon Resource Name (ARN) 値は、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 **encypted** 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。

10.2.5. 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ルするのに十分なストレージがボリュームプロ

ビンョニンクされます。

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 はマネージド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マネージド外ディスクの両方の使用を許可しないため、**Shared** または **Dedicated** で作成されたマネージド外ディスクを 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。

10.2.6. 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。

10.2.6.1. 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、コンテナーのプロセス 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。

10.2.7. 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。

10.2.8. 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> 1
provisioner: csi.vsphere.vmware.com 2
```

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 OpenShift Container Platform での VMware vSphere CSI の使用の詳細は、Kubernetes のドキュメント を参照してください。

10.3. デフォルトストレージクラスの変更

次の手順を使用して、デフォルトのストレージクラスを変更します。

たとえば、**gp3** 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 **gp3** から **standard** に変更する必要がある場合などです。

前提条件

- クラスター管理者権限でクラスターにアクセスできる。

手順

デフォルトのストレージクラスを変更するには、以下を実行します。

1. ストレージクラスを一覧表示します。

```
$ oc get storageclass
```

出力例

```
NAME            TYPE
gp3 (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. 目的のストレージクラスをデフォルトにします。
目的のストレージクラスに、次のコマンドを実行して **storageclass.kubernetes.io/is-default-class** アノテーションを **true** に設定します。

```
$ oc patch storageclass standard -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

注記

短期間であれば、複数のデフォルトのストレージクラスを使用できます。ただし、最終的には 1 つのデフォルトのストレージクラスのみが存在することを確認する必要があります。

複数のデフォルトストレージクラスが存在する場合、デフォルトストレージクラス (**pvc.spec.storageClassName** =nil) を要求するすべての永続ボリューム要求 (PVC) は、そのストレージクラスのデフォルトステータスと管理者に関係なく、最後に作成されたデフォルトストレージクラスを取得します。アラートダッシュボードで、複数のデフォルトストレージクラス **MultipleDefaultStorageClasses** があるというアラートを受け取ります。

3. 古いデフォルトストレージクラスからデフォルトのストレージクラス設定を削除します。古いデフォルトのストレージクラスの場合は、次のコマンドを実行して **storageclass.kubernetes.io/is-default-class** アノテーションの値を **false** に変更します。

```
$ oc patch storageclass gp3 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

4. 変更内容を確認します。

```
$ oc get storageclass
```

出力例

```
NAME                TYPE
gp3                 kubernetes.io/aws-ebs
standard (default)  kubernetes.io/aws-ebs
```

10.4. 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。

10.5. 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10.1 利用可能なストレージオプション

ストレージタイプ	説明	例

ストレージタイプ	説明	例
ブロック	• ブロックデバイスとしてオペレーティングシステムに公開されます。 • ストレージを完全に制御し、ファイルシステムを通過してファイルの低いレベルで操作する必要のあるアプリケーションに適しています。 • ストレージエリアネットワーク (SAN) とも呼ばれます。 • 共有できません。 一度に 1 つのクライアントだけがこのタイプのエンドポイントをマウントできるという意味です。	AWS EBS および VMware vSphere は、OpenShift Container Platform で永続ボリューム (PV) の動的なプロビジョニングをサポートします。
ファイル	• マウントされるファイルシステムのエクスポートとして、OS に公開されます。 • ネットワークアタッチストレージ (NAS) とも呼ばれます。 • 同時実行、レイテンシー、ファイルロックのメカニズムその他の各種機能は、プロトコルおよび実装、ベンダー、スケールによって大きく異なります。	RHEL NFS、NetApp NFS [1]、および Vendor NFS
オブジェクト	• REST API エンドポイント経由でアクセスできます。 • OpenShift 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ー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。

10.6. 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10.2 設定可能な推奨ストレージ技術

ストレージタイプ	ブロック	ファイル	オブジェクト
ROX[1]	はい[4]	はい[4]	はい
RWX[2]	いいえ	はい	はい

| ストレージタイプ | ブロック | ファイル | オブジェクト |

レジストリー	設定可能	設定可能	推奨
スケーリングされたレジストリー	設定不可	設定可能	推奨
メトリクス[3]	推奨	設定可能[5]	設定不可
Elasticsearch ロギング	推奨	設定可能[6]	サポート対象外[6]
Loki ロギング	設定不可	設定不可	推奨
アプリ	推奨	推奨	設定不可[7]

[1] **ReadOnlyMany**

[2] **ReadWriteMany**

[3] **Prometheus はメトリック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] **メトリックの場合、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。**

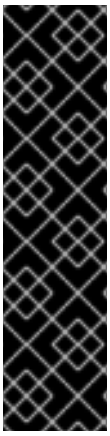
[6] **ログについては、ログストアの永続ストレージの設定セクションで推奨されるストレージソリューションを確認してください。NFS ストレージを永続ボリュームとして使用するか、Gluster などの NAS を介して使用すると、データが破損する可能性があります。したがって、NFS は、OpenShift Container Platform Logging の Elasticsearch ストレージおよび LokiStack ログストアではサポートされていません。ログストアごとに 1 つの永続的なボリュームタイプを使用する必要があります。**

[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。

10.6.1. 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、メトリックストレージの Prometheus、およびロギングストレージの Elasticsearch が含まれます。そのため、コアサービスで使用される PV をサポートするために RHEL NFS を使用することは推奨されていません。

他の NFS の実装ではこれらの問題が検出されない可能性があります。OpenShift Container Platform コアコンポーネントに対して実施された可能性のあるテストに関する詳細情報は、個別の NFS 実装ベンダーにお問い合わせください。

10.6.1.1. 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。

10.6.1.2. 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。

10.6.1.3. メトリクス

OpenShift Container Platform がホストするメトリック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ッククラスターデプロイメントにファイルストレージを使用することは推奨されません。

10.6.1.4. 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:

- Loki Operator:
  - 推奨されるストレージテクノロジーは、S3 互換のオブジェクトストレージです。
  - ブロックストレージは設定できません。
- OpenShift Elasticsearch Operator:
  - 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サポートされていません。

注記

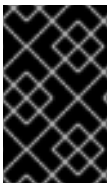
Logging バージョン 5.4.3 の時点で、OpenShift Elasticsearch Operator は非推奨であり、今後のリリースで削除される予定です。Red Hat は、この機能に対して現在のリリースライフサイクル中にバグ修正とサポートを提供しますが、拡張機能の提供はなく、この機能は今後削除される予定です。OpenShift Elasticsearch Operator を使用してデフォルトのログストレージを管理する代わりに、Loki Operator を使用できます。

10.6.1.5. アプリケーション

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

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

10.6.2. 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推奨しません。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

10.7. RED HAT OPENSHIFT 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 4.12 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、スケーリングに関する推奨事項	OpenShift Data Foundation 4.12 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 4.12 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 4.12 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 への OpenShift Data Foundation 4.12 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 Services を使用した OpenShift Data Foundation 4.12 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 4.12 のデプロイおよび管理

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
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 4.12 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ように OpenShift Data Foundation をデプロイする手順	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ように OpenShift Data Foundation をデプロイする手順	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
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 4.12 クラスターのモニタリング	Red Hat OpenShift Data Foundation 4.12 のモニタリング
操作中に発生する問題の解決	OpenShift Data Foundation 4.12 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

10.8. 関連情報

- Elasticsearch ログストアの設定

第11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。

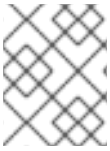
11.1. アイデンティティープロバイダー設定について

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ープロバイダーを指定するように設定できます。

11.1.1. OpenShift Container Platform のアイデンティティー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。アイデンティティープロバイダーを指定するには、アイデンティティー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。

11.1.2. サポートされるアイデンティティープロバイダー

以下の種類のアイデンティティープロバイダーを設定できます。

アイデンティティープロバイダー	説明
htpasswd	**htpasswd** アイデンティティープロバイダーを **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** アイデンティティープロバイダーを、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** アイデンティティープロバイダーを、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** アイデンティティープロバイダーを、ユーザーがリモートアイデンティティー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** アイデンティティープロバイダーを、**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ーと併用されます。

アイデンティティープロバイダー	説明
GitHub または GitHub Enterprise	**github** アイデンティティープロバイダーを、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** アイデンティティープロバイダーを、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ープロバイダーとして使用するように設定します。
Google	**google** アイデンティティープロバイダーを、Google の OpenID Connect 統合を使用して設定します。
OpenID Connect	**oidc** アイデンティティープロバイダーを、Authorization Code Flow を使用して OpenID Connect アイデンティティープロバイダーと統合するように設定します。

アイデンティティー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。

11.1.3. アイデンティティー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ー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、アイデンティティー名が作成されます。
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。 **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。

11.1.4. 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 1
    mappingMethod: claim 2
    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、アイデンティティー名が作成されます。
2. このプロバイダーのアイデンティティーと **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。

11.2. 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。

11.2.1. 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。認可は認証とは異なる手順であることに注意してください。認証はアクションを実行するユーザーのアイデンティティー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。
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。

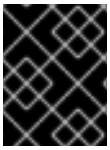
この 2 つのレベルからなる階層により、ローカルロールで個別プロジェクト内のカスタマイズが可能になる一方で、クラスターロールによる複数プロジェクト間での再利用が可能になります。

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。

11.2.1.1. 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。



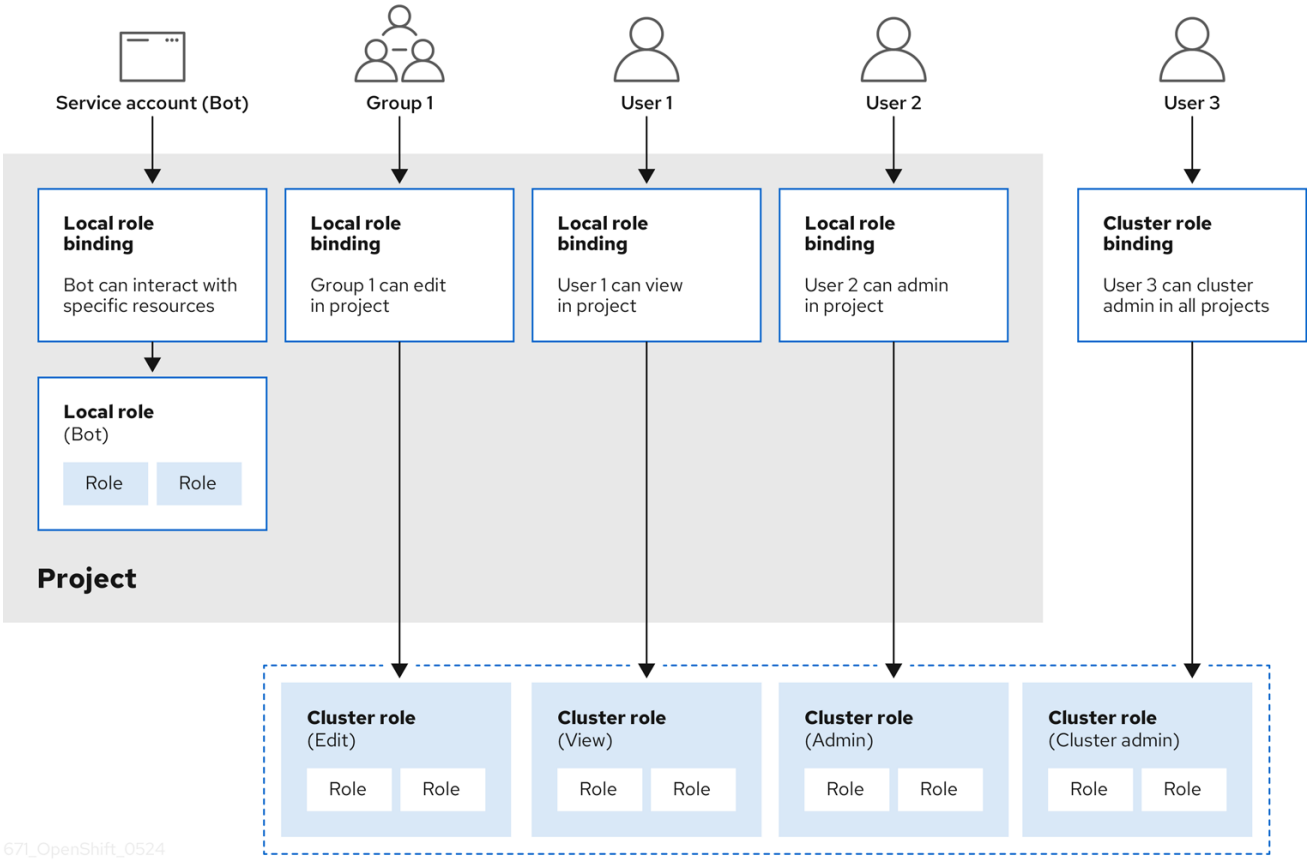
重要

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使用される場合、**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は、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。
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リスト表示しない 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リスト表示します。

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。

11.2.1.2. 認可の評価

OpenShift Container Platform は以下を使用して認可を評価します。

アイデンティティー

ユーザーが属するユーザー名とグループのリスト。

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ーは、他のコミュニティー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リスト、ロールを持つユーザーまたはグループ間の関連付け。

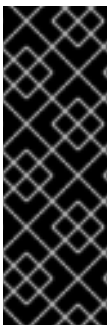
OpenShift Container Platform は以下の手順を使用して認可を評価します。

1. アイデンティティーおよびプロジェクトでスコープ設定されたアクションは、ユーザーおよびそれらのグループに適用されるすべてのバインディングを検索します。
2. バインディングは、適用されるすべてのロールを見つけるために使用されます。
3. ロールは、適用されるすべてのルールを見つけるために使用されます。
4. 一致を見つけるために、アクションが各ルールに対してチェックされます。
5. 一致するルールが見つからない場合、アクションはデフォルトで拒否されます。

ヒント

ユーザーおよびグループは一度に複数のロールに関連付けたり、バインドしたりできることに留意してください。

プロジェクト管理者は CLI を使用してローカルロールとローカルバインディングを表示できます。これには、それぞれのロールが関連付けられる動詞およびリソースのマトリクスが含まれます。



重要

プロジェクト管理者にバインドされるクラスターロールは、ローカルバインディングによってプロジェクト内で制限されます。これは、**cluster-admin** または **system:admin** に付与されるクラスターロールのようにクラスター全体でバインドされる訳ではありません。

クラスターロールは、クラスターレベルで定義されるロールですが、クラスターレベルまたはプロジェクトレベルのいずれかでバインドできます。

11.2.1.2.1. 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。

11.2.2. 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。
- コミュニティー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、namaspace 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ーが他のコミュニティー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の識別子であり、CLI ツールまたは API を使用する場合に最も表示されます。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。

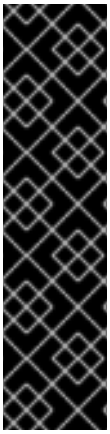
オブジェクト	説明
Objects	Pod、サービス、レプリケーションコントローラーなど。
Policies	ユーザーがオブジェクトに対してアクションを実行できるか、できないかについてのルール。
Constraints	制限を設定できるそれぞれの種類のオブジェクトのクォータ。
Service accounts	サービスアカウントは、プロジェクトのオブジェクトへの指定されたアクセスで自動的に機能します。

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ー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。

11.2.3. 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 によるアドミッションが保証されます。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。



重要

デフォルトプロジェクトでワークロードを実行したり、デフォルトプロジェクトへのアクセスを共有したりしないでください。デフォルトのプロジェクトは、コアクラスターコンポーネントを実行するために予約されています。

デフォルトプロジェクトである **default**、**kube-public**、**kube-system**、**openshift**、**openshift-infra**、**openshift-node**、および **openshift.io/run-level** ラベルが **0** または **1** に設定されているその他のシステム作成プロジェクトは、高い特権があるとみなされます。Pod セキュリティーアドミッション、セキュリティーコンテキスト制約、クラスターリソースクォータ、イメージ参照解決などのアドミッションプラグインに依存する機能は、高い特権を持つプロジェクトでは機能しません。

11.2.4. 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]
  pods/exec                                                  []              []              [get list watch create delete
deletecollection patch update]
  pods/portforward                                           []              []              [get list watch create
delete deletecollection patch update]
  pods/proxy                                                 []              []              [get list watch create delete
deletecollection patch update]
  services/proxy                                             []              []              [get list watch create delete
deletecollection patch update]
  routes/status                                              []              []              [get list watch update]
  routes.route.openshift.io/status                           []              []              [get list watch update]
  appliedclusterresourcequotas                               []              []              [get list watch]
  bindings                                                   []              []              [get list watch]
  builds/log                                                 []              []              [get list watch]
  deploymentconfigs/log                                      []              []              [get list watch]
  deploymentconfigs/status                                   []              []              [get list watch]
  events                                                     []              []              [get list watch]
  imagestreams/status                                        []              []              [get list watch]
  limitranges                                                []              []              [get list watch]
  namespaces/status                                          []              []              [get list watch]
  pods/log                                                   []              []              [get list watch]
  pods/status                                                []              []              [get list watch]
  replicationcontrollers/status                              []              []              [get list watch]
  resourcequotas/status                                      []              []              [get list watch]
  resourcequotas                                             []              []              [get list watch]
  resourcequotausages                                        []              []              [get list watch]
  rolebindingrestrictions                                    []              []              [get list watch]
  deploymentconfigs.apps.openshift.io/log                    []              []              [get list watch]
  deploymentconfigs.apps.openshift.io/status                 []              []              [get list watch]
  controllerrevisions.apps                                   []              []              [get list watch]
  rolebindingrestrictions.authorization.openshift.io         []              []              [get list watch]
  builds.build.openshift.io/log                              []              []              [get list watch]
  imagestreams.image.openshift.io/status                     []              []              [get list watch]
  appliedclusterresourcequotas.quota.openshift.io            []              []              [get list watch]
  imagestreams/layers                                        []              []              [get update get]
  imagestreams.image.openshift.io/layers                     []              []              [get update get]
  builds/details                                             []              []              [update]
  builds.build.openshift.io/details                          []              []              [update]


Name:         basic-user
Labels:       <none>
```

```
Annotations:  openshift.io/description: A user that can get basic information about projects.
              rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
 Resources                                  Non-Resource URLs  Resource Names  Verbs
 ---------                                  -----------------  --------------  -----
 selfsubjectrulesreviews                    []                 []              [create]
 selfsubjectaccessreviews.authorization.k8s.io  []             []              [create]
 selfsubjectrulesreviews.authorization.openshift.io  []         []              [create]
 clusterroles.rbac.authorization.k8s.io     []                 []              [get list watch]
 clusterroles                               []                 []              [get list]
 clusterroles.authorization.openshift.io    []                 []              [get list]
 storageclasses.storage.k8s.io              []                 []              [get list]
 users                                      []                 [~]             [get]
 users.user.openshift.io                    []                 [~]             [get]
 projects                                   []                 []              [list watch]
 projects.project.openshift.io              []                 []              [list watch]
 projectrequests                            []                 []              [list]
 projectrequests.project.openshift.io       []                 []              [list]

Name:         cluster-admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
Resources  Non-Resource URLs  Resource Names  Verbs
---------  -----------------  --------------  -----
*.*        []                 []              [*]
           [*]                []              [*]

...
```

2. 各種のロールにバインドされたユーザーおよびグループを示す、クラスターのロールバインディングの現在のセットを表示するには、以下を実行します。

```
$ oc describe clusterrolebinding.rbac
```

出力例

```
Name:         alertmanager-main
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  alertmanager-main
Subjects:
  Kind           Name              Namespace
  ----           ----              ---------
  ServiceAccount  alertmanager-main  openshift-monitoring


Name:         basic-users
Labels:       <none>
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  basic-user
```

```
Subjects:
  Kind   Name                Namespace
  ----   ----                ---------
  Group  system:authenticated


Name:         cloud-credential-operator-rolebinding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  cloud-credential-operator-role
Subjects:
  Kind            Name     Namespace
  ----            ----     ---------
  ServiceAccount  default  openshift-cloud-credential-operator


Name:         cluster-admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  cluster-admin
Subjects:
  Kind   Name            Namespace
  ----   ----            ---------
  Group  system:masters


Name:         cluster-admins
Labels:       <none>
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  cluster-admin
Subjects:
  Kind   Name                  Namespace
  ----   ----                  ---------
  Group  system:cluster-admins
  User   system:admin


Name:         cluster-api-manager-rolebinding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  cluster-api-manager-role
Subjects:
  Kind            Name     Namespace
  ----            ----     ---------
  ServiceAccount  default  openshift-machine-api

...
```

11.2.5. 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


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
```

11.2.6. 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
```

2. 出力でローカルロールバインディングを確認し、追加の内容を確認します。

```
$ oc describe rolebinding.rbac -n <project>
```

たとえば、**joe** プロジェクトのローカルロールバインディングを表示するには、以下を実行します。

```
$ oc describe rolebinding.rbac -n joe
```

出力例

```
Name:         admin
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  admin
Subjects:
  Kind  Name        Namespace
  ----  ----        ---------
  User  kube:admin


Name:         admin-0
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  admin
Subjects:
  Kind  Name   Namespace
  ----  ----   ---------
  User  alice (1)


Name:         system:deployers
```

```
Labels:       <none>
Annotations:  openshift.io/description:
              Allows deploymentconfigs in this namespace to rollout pods in
              this namespace.  It is auto-managed by a controller; remove
              subjects to disa...
Role:
  Kind:  ClusterRole
  Name:  system:deployer
Subjects:
  Kind           Name      Namespace
  ----           ----      ---------
  ServiceAccount  deployer  joe


Name:         system:image-builders
Labels:       <none>
Annotations:  openshift.io/description:
              Allows builds in this namespace to push images to this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-builder
Subjects:
  Kind           Name     Namespace
  ----           ----     ---------
  ServiceAccount  builder  joe


Name:         system:image-pullers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows all pods in this namespace to pull images from this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-puller
Subjects:
  Kind   Name                         Namespace
  ----   ----                         ---------
  Group  system:serviceaccounts:joe
```

1. **alice** ユーザーが **admins RoleBinding** に追加されています。

11.2.7. 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リストです。
- **<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
```

11.2.8. 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リストです。
- **<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
```

11.2.9. 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11.1 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、そのすべてのロールを削除します。
$ 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、そのすべてのロールを削除します。

11.2.10. 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11.2 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。

11.2.11. 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

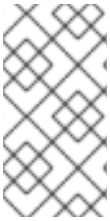
- クラスター管理者として定義するユーザーを作成している。

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>
```

11.2.12. 認証されていないグループのクラスターロールバインディング



注記

OpenShift Container Platform 4.16 より前では、認証されていないグループでも一部のクラスターロールへのアクセスが許可されていました。OpenShift Container Platform 4.16 より前のバージョンから更新されたクラスターは、認証されていないグループに対してこのアクセスを保持します。

セキュリティー上の理由から、OpenShift Container Platform 4.16 では、デフォルトで、認証されていないグループはクラスターロールにアクセスできません。

ユースケースによっては、クラスターロールに **system:unauthenticated** を追加する必要があります。

クラスター管理者は、認証されていないユーザーを次のクラスターロールに追加できます。

- **system:scope-impersonation**
- **system:webhook**
- **system:oauth-token-deleter**
- **self-access-reviewer**

重要

認証されていないアクセスを変更するときは、常に組織のセキュリティー標準に準拠していることを確認してください。

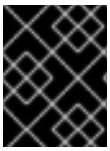
11.2.13. 認証されていないグループのクラスターロールへの追加

クラスター管理者は、クラスターロールバインディングを作成することにより、OpenShift Container Platform の次のクラスターロールに認証されていないユーザーを追加できます。認証されていないユーザーには、パブリックではないクラスターロールへのアクセス権はありません。これは、特定のユースケースで必要な場合にのみ行うようにしてください。

認証されていないユーザーを以下のクラスターロールに追加できます。

- **system:scope-impersonation**
- **system:webhook**
- **system:oauth-token-deleter**

- **self-access-reviewer**



重要

認証されていないアクセスを変更するときは、常に組織のセキュリティー標準に準拠していることを確認してください。

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

手順

1. **add-<cluster_role>-unauth.yaml** という名前の YAML ファイルを作成し、次のコンテンツを追加します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 annotations:
   rbac.authorization.kubernetes.io/autoupdate: "true"
 name: <cluster_role>access-unauthenticated
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: <cluster_role>
subjects:
 - apiGroup: rbac.authorization.k8s.io
   kind: Group
   name: system:unauthenticated
```

2. 以下のコマンドを実行して設定を適用します。

```
$ oc apply -f add-<cluster_role>.yaml
```

11.3. KUBEADMIN ユーザー

OpenShift Container Platform は、インストールプロセスの完了後にクラスター管理者 **kubeadmin** を作成します。

このユーザーには、**cluster-admin** ロールが自動的に適用され、このユーザーはクラスターの root ユーザーとしてみなされます。パスワードは動的に生成され、OpenShift Container Platform 環境に対して一意です。インストールの完了後に、パスワードはインストールプログラムの出力で提供されます。以下に例を示します。

```
INFO Install complete!
INFO Run 'export KUBECONFIG=<your working directory>/auth/kubeconfig' to manage the cluster
with 'oc', the OpenShift CLI.
INFO The cluster is ready when 'oc login -u kubeadmin -p <provided>' succeeds (wait a few minutes).
INFO Access the OpenShift web-console here: https://console-openshift-
console.apps.demo1.openshift4-beta-abcorp.com
INFO Login to the console with user: kubeadmin, password: <provided>
```

11.3.1. kubeadmin ユーザーの削除

アイデンティティープロバイダーを定義し、新規 **cluster-admin** ユーザーを作成した後に、クラスターのセキュリティーを強化するために **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ー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
```

11.4. 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。

11.4.1. 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

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ーイメージレジストリーを制限します。このリストを、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 config map の参照です。 この config map の namespace は **openshift-config** です。config map 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。
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。

パラメーター	説明
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ーを指定できます。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ーを指定できます。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ーを指定できます。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 Image Registry 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、**OPENSHIFT_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	Image Registry Operator によって設定され、イメージレジストリーが外部に公開されるときに、イメージレジストリーの外部ホスト名を提供します。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。

11.4.2. 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: (2)
    - domainName: quay.io
```

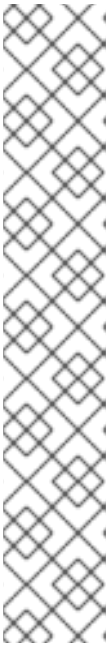
```
    insecure: false
  additionalTrustedCA: 3
    name: myconfigmap
  registrySources: 4
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - image-registry.openshift-image-registry.svc:5000
    - reg1.io/myrepo/myapp:latest
    insecureRegistries:
    - insecure.com
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1 **Image**: イメージの処理方法についてのクラスター全体の情報を保持します。正規名および唯一の有効な名前となるのは **cluster** です。

2 **allowedRegistriesForImport**: 標準ユーザーがイメージのインポートに使用するコンテナーイメージレジストリーを制限します。このリストを、有効なイメージを含むものとしてユーザーが信頼し、アプリケーションのインポート元となるレジストリーに設定します。イメージまたは **ImageStreamMappings** を API 経由で作成するパーミッションを持つユーザーは、このポリシーによる影響を受けません。通常、これらのパーミッションを持っているのはクラスター管理者のみです。

3 **additionalTrustedCA**: イメージストリームのインポート、Pod のイメージプル、**openshift-image-registry** プルスルー、およびビルド時に信頼される追加の認証局 (CA) が含まれる config map の参照です。この config map の namespace は **openshift-config** です。config map の形式では、信頼する追加のレジストリー CA についてレジストリーのホスト名をキーとして使用し、PEM 証明書を値として使用します。

4 **registrySources**: ビルドおよび Pod のイメージにアクセスする際に、コンテナー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ーを指定できます。例: **reg1.io/myrepo/myapp:latest**

セキュリティー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   v1.29.4
ip-10-0-139-120.us-east-2.compute.internal   Ready,SchedulingDisabled   control-plane          74m   v1.29.4
ip-10-0-176-102.us-east-2.compute.internal   Ready                      control-plane          75m   v1.29.4
ip-10-0-188-96.us-east-2.compute.internal    Ready                      worker                 65m   v1.29.4
ip-10-0-200-59.us-east-2.compute.internal    Ready                      worker                 63m   v1.29.4
ip-10-0-223-123.us-east-2.compute.internal   Ready                      control-plane          73m   v1.29.4
```

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。

11.4.3. イメージレジストリーアクセス用の追加のトラストストアの設定

image.config.openshift.io/cluster カスタムリソースには、イメージレジストリーのアクセス時に信頼される追加の認証局が含まれる 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

openshift-config namespace で config map を作成し、**image.config.openshift.io** カスタムリソースの **AdditionalTrustedCA** でその名前を使用して、外部レジストリーにアクセスするときに信頼する必要がある追加の CA を提供できます。

config map のキーは、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: | 1
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

1 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。

- 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
```

11.5. イメージレジストリーリポジトリーのミラーリングについて

コンテナーレジストリーリポジトリーのミラーリングを設定すると、次のタスクを実行できます。

- ソースイメージのレジストリーのリポジトリー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ーで解決できるようにします。
- 各ターゲットリポジトリーに対して複数のミラーリングされたリポジトリーを特定し、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。

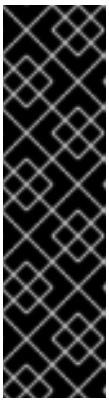
OpenShift Container Platform のリポジトリーミラーリングには、以下の属性が含まれます。

- 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。

- 非接続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ー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ー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ー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。

リポジトリー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 に必要なコンテナーイメージをプルし、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、非接続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 のインストール中にミラーリングを設定しなかった場合は、以下のカスタムリソース (CR) オブジェクトのいずれかを使用して、インストール後に設定できます。
  - **ImageDigestMirrorSet** (IDMS)。このオブジェクトを使用すると、ダイジェスト仕様を使用して、ミラーリングされたレジストリーからイメージを取得できます。IDMS CR を使用すると、イメージのプルが失敗した場合に、ソースレジストリーからのプルの継続的な試行を許可または停止するフォールバックポリシーを設定できます。
  - **ImageTagMirrorSet** (ITMS)。このオブジェクトを使用すると、イメージタグを使用して、ミラーリングされたレジストリーからイメージをプルできます。ITMS CR を使用すると、イメージのプルが失敗した場合に、ソースレジストリーからのプルの継続的な試行を許可または停止するフォールバックポリシーを設定できます。
  - **ImageContentSourcePolicy** (ICSP)。このオブジェクトを使用すると、ダイジェスト仕様を使用して、ミラーリングされたレジストリーからイメージを取得できます。ミラーが機能しない場合、ICSP CR は必ずソースレジストリーにフォールバックします。



重要

ImageContentSourcePolicy (ICSP) オブジェクトを使用してリポジトリーミラーリングを設定することは、非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。**ImageContentSourcePolicy** オブジェクトの作成に使用した既存の YAML ファイルがある場合は、**oc adm migrate icsp** コマンドを使用して、それらのファイルを **ImageDigestMirrorSet** YAML ファイルに変換できます。詳細は、次のセクションの「イメージレジストリーリポジトリーミラーリング用の ImageContentSourcePolicy (ICSP) ファイルの変換」を参照してください。

これらのカスタムリソースオブジェクトはそれぞれ、次の情報を識別します。

- ミラーリングするコンテナーイメージリポジトリーのソース

- ソースリポジトリー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ーの個別のエントリー。

新しいクラスターの場合は、必要に応じて IDMS、ITMS、および ICSP CR オブジェクトを使用できます。ただし、IDMS と ITMS の使用を推奨します。

クラスターをアップグレードした場合、既存の ICSP オブジェクトは安定を維持し、IDMS オブジェクトと ICSP オブジェクトの両方がサポートされるようになります。ICSP オブジェクトを使用するワークロードは、引き続き期待どおりに機能します。一方、IDMS CR で導入されたフォールバックポリシーを利用する場合は、**oc adm merge icsp** コマンドを使用して、現在のワークロードを IDMS オブジェクトに移行できます。これについては、後述の **イメージレジストリーリポジトリーミラーリング用の ImageContentSourcePolicy (ICSP) ファイルの変換** セクションで説明しています。IDMS オブジェクトへの移行に、クラスターの再起動は必要ありません。

注記

クラスターで **ImageDigestMirrorSet**、**ImageTagMirrorSet**、または **ImageContentSourcePolicy** オブジェクトを使用してリポジトリーミラーリングを設定する場合、ミラーリングされたレジストリーにはグローバルプルシークレットのみを使用できます。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。

11.5.1. イメージレジストリーのリポジトリーミラーリングの設定

インストール後のミラー設定カスタムリソース (CR) を作成して、ソースイメージレジストリーからミラーリングされたイメージレジストリーにイメージプル要求をリダイレクトできます。

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. ミラーリングされたリポジトリーを設定します。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リポジトリーを設定します。Red Hat Quay を使用すると、あるリポジトリーから別のリポジトリーにイメージをコピーでき、これらのリポジトリー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、ソースリポジトリーからミラーリングされたリポジトリー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/ubi9/ubi-minimal:latest@sha256:5cf... \
     docker://example.io/example/ubi-minimal
     ```

     この例では、**example.io** いう名前のコンテナー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ーがあり、そこに **registry.access.redhat.com** から **ubi9/ubi-minimal** イメージをコピーします。ミラーリングされたレジストリーを作成した後、ソースリポジトリーに対する要求をミラーリングされたリポジトリーにリダイレクトするように OpenShift Container Platform クラスターを設定できます。

2. OpenShift Container Platform クラスターにログインします。

3. 次の例のいずれかを使用して、インストール後のミラー設定 CR を作成します。

- 必要に応じて **ImageDigestMirrorSet** または **ImageTagMirrorSet** CR を作成し、ソースとミラーを独自のレジストリーとリポジトリーのペアとイメージに置き換えます。

```
apiVersion: config.openshift.io/v1 1
kind: ImageDigestMirrorSet 2
metadata:
  name: ubi9repo
spec:
  imageDigestMirrors: 3
  - mirrors:
    - example.io/example/ubi-minimal 4
    - example.com/example/ubi-minimal 5
    source: registry.access.redhat.com/ubi9/ubi-minimal 6
    mirrorSourcePolicy: AllowContactingSource 7
  - mirrors:
    - mirror.example.com/redhat
    source: registry.example.com/redhat 8
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.com
    source: registry.example.com 9
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.net/image
    source: registry.example.com/example/myimage 10
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.net
    source: registry.example.com/example 11
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.net/registry-example-com
    source: registry.example.com 12
    mirrorSourcePolicy: AllowContactingSource
```

1. この CR で使用する API を示します。これは **config.openshift.io/v1** である必要があります。

2. プルタイプに応じてオブジェクトの種類を示します。
   - **ImageDigestMirrorSet**: ダイジェスト参照イメージをプルします。
   - **ImageTagMirrorSet**: タグ参照イメージをプルします。

3. 次のいずれかのイメージプルメソッドのタイプを示します。
   - **imageDigestMirrors**: **ImageDigestMirrorSet** CR に使用します。
   - **imageTagMirrors**: **ImageTagMirrorSet** CR に使用します。

4. ミラーリングされたイメージのレジストリーとリポジトリーの名前を示します。

5 オプション: 各ターゲットリポジトリーのセカンダリーミラーリポジトリーを示します。1 つのミラーがダウンした場合、ターゲットリポジトリーは別のミラーを使用でき

6 イメージプル仕様で参照されるリポジトリーである、レジストリーおよびリポジトリーソースを示します。

7 オプション: イメージのプルが失敗した場合のフォールバックポリシーを示します。

- **AllowContactingSource**: ソースリポジトリーからのイメージのプルの継続的な試行を許可します。これはデフォルトになります。
- **NeverContactSource**: ソースリポジトリーからのイメージのプルの継続的な試行を防ぎます。

8 オプション: レジストリー内の namespace を示します。これにより、その namaspace で任意のイメージを使用できます。レジストリードメインをソースとして使用する場合、オブジェクトはレジストリーからすべてのリポジトリーに適用されます。

9 オプション: レジストリーを示し、そのレジストリー内の任意のイメージを使用できるようにします。レジストリー名を指定すると、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ーにオブジェクトが適用されます。

10 イメージ **registry.example.com/example/myimage@sha256:...** をミラー **mirror.example.net/image@sha256:..** からプルします。

11 ミラー **mirror.example.net/image@sha256:...** からソースレジストリー namespace のイメージ **registry.example.com/example/image@sha256:...** をプルします。

12 ミラーレジストリー **example.net/registry-example-com/myimage@sha256:...** からイメージ **registry.example.com/myimage@sha256** をプルします。

- **ImageContentSourcePolicy** カスタムリソースを作成し、ソースとミラーを独自のレジストリーとリポジトリーのペアとイメージに置き換えます。

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: mirror-ocp
spec:
  repositoryDigestMirrors:
  - mirrors:
    - mirror.registry.com:443/ocp/release 1
    source: quay.io/openshift-release-dev/ocp-release 2
  - mirrors:
    - mirror.registry.com:443/ocp/release
    source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

1 ミラーイメージレジストリーおよびリポジトリーの名前を指定します。

2 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ーを指定します。

4. 新規オブジェクトを作成します。

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

オブジェクトの作成後、Machine Config Operator (MCO) は **ImageTagMirrorSet** オブジェクトのみのノードをドレインします。MCO は、**ImageDigestMirrorSet** オブジェクトと **ImageContentSourcePolicy** オブジェクトのノードをドレインしません。

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   v1.29.4
   ip-10-0-138-148.ec2.internal   Ready                         master   11m  v1.29.4
   ip-10-0-139-122.ec2.internal   Ready                         master   11m  v1.29.4
   ip-10-0-147-35.ec2.internal    Ready                        worker   7m   v1.29.4
   ip-10-0-153-12.ec2.internal    Ready                        worker   7m   v1.29.4
   ip-10-0-154-10.ec2.internal    Ready                        master   11m  v1.29.4
   ```

   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
   ```

   次の出力は、インストール後のミラー設定 CR が適用された **registries.conf** ファイルを表しています。最後の 2 つのエントリーは、それぞれ **digest-only** および **tag-only** とマークされています。

   出力例

   ```
   unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
   short-name-mode = ""

   [[registry]]
     prefix = ""
     location = "registry.access.redhat.com/ubi9/ubi-minimal" (1)
   ```

```
  [[registry.mirror]]
    location = "example.io/example/ubi-minimal" (2)
    pull-from-mirror = "digest-only" (3)

  [[registry.mirror]]
    location = "example.com/example/ubi-minimal"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com"

  [[registry.mirror]]
    location = "mirror.example.net/registry-example-com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com/example"

  [[registry.mirror]]
    location = "mirror.example.net"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com/example/myimage"

  [[registry.mirror]]
    location = "mirror.example.net/image"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com"

  [[registry.mirror]]
    location = "mirror.example.com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com/redhat"

  [[registry.mirror]]
    location = "mirror.example.com/redhat"
    pull-from-mirror = "digest-only"
[[registry]]
  prefix = ""
  location = "registry.access.redhat.com/ubi9/ubi-minimal"
  blocked = true (4)

  [[registry.mirror]]
    location = "example.io/example/ubi-minimal-tag"
    pull-from-mirror = "tag-only" (5)
```

```
```

1. プルスペックで参照されるリポジトリーを示します。
2. そのリポジトリーのミラーを示します。
3. ミラーからプルされたイメージがダイジェスト参照イメージであることを示します。
4. このリポジトリーに **NeverContactSource** パラメーターが設定されていることを示します。
5. ミラーからプルされたイメージがタグ参照イメージであることを示します。

e. ソースからノードにイメージをプルし、ミラーによって解決されるかどうかを確認します。

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi9/ubi-minimal@sha256:5cf...
```

リポジトリーのミラーリングのトラブルシューティング

リポジトリーのミラーリング手順が説明どおりに機能しない場合は、リポジトリー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。

11.5.2. イメージレジストリーリポジトリーミラーリング用の ImageContentSourcePolicy (ICSP) ファイルの変換

ImageContentSourcePolicy (ICSP) オブジェクトを使用してリポジトリーミラーリングを設定することは、非推奨の機能です。この機能は引き続き OpenShift Container Platform に含まれており、引き続きサポートされます。ただし、この製品の将来のリリースでは削除される予定であり、新しいデプロイメントには推奨されません。

ICSP オブジェクトは、リポジトリーミラーリングを設定するために **ImageDigestMirrorSet** および **ImageTagMirrorSet** オブジェクトに置き換えられています。**ImageContentSourcePolicy** オブジェクトの作成に使用した既存の YAML ファイルがある場合は、**oc adm migrate icsp** コマンドを使用して、それらのファイルを **ImageDigestMirrorSet** YAML ファイルに変換できます。このコマンドは、API を現在のバージョンに更新し、**kind** 値を **ImageDigestMirrorSet** に変更し、**spec.repositoryDigestMirrors** を **spec.imageDigestMirrors** に変更します。ファイルの残りの部分は変更されません。

移行によって **registries.conf** ファイルは変更されないため、クラスターを再起動する必要はありません。

ImageDigestMirrorSet または **ImageTagMirrorSet** オブジェクトの詳細については、前のセクションの「イメージレジストリーリポジトリーミラーリングの設定」を参照してください。

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。
- クラスターに **ImageContentSourcePolicy** オブジェクトがあることを確認します。

手順

1. 次のコマンドを使用して、1 つ以上の **ImageContentSourcePolicy** YAML ファイルを **ImageDigestMirrorSet** YAML ファイルに変換します。

```
$ oc adm migrate icsp <file_name>.yaml <file_name>.yaml <file_name>.yaml --dest-dir
<path_to_the_directory>
```

ここでは、以下のようになります。

<file_name>

ソース **ImageContentSourcePolicy** YAML の名前を指定します。複数のファイル名をリストできます。

--dest-dir

オプション: 出力 **ImageDigestMirrorSet** YAML のディレクトリーを指定します。設定されていない場合、ファイルは現在のディレクトリーに書き込まれます。

たとえば、次のコマンドは **icsp.yaml** および **icsp-2.yaml** ファイルを変換し、新しい YAML ファイルを **idms-files** ディレクトリーに保存します。

```
$ oc adm migrate icsp icsp.yaml icsp-2.yaml --dest-dir idms-files
```

出力例

```
wrote ImageDigestMirrorSet to idms-
files/imagedigestmirrorset_ubi8repo.5911620242173376087.yaml
wrote ImageDigestMirrorSet to idms-
files/imagedigestmirrorset_ubi9repo.6456931852378115011.yaml
```

2. 次のコマンドを実行して CR オブジェクトを作成します。

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

ここでは、以下のようになります。

<path_to_the_directory>

--dest-dir フラグを使用した場合は、ディレクトリーへのパスを指定します。

<file_name>

ImageDigestMirrorSet YAML の名前を指定します。

3. IDMS オブジェクトがロールアウトされた後、ICSP オブジェクトを削除します。

11.6. 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。

11.6.1. 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

11.6.2. 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で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 ディレクトリーで **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。

11.6.3. 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。

ヒント

または、Web コンソールを使用してカタログソースを管理できます。**Administration → Cluster Settings → Configuration → OperatorHub** ページから、**Sources** タブをクリックして、個別のソースを作成、更新、削除、無効化、有効化できます。

前提条件

- インデックスイメージをビルドしてレジストリーにプッシュしている。
- **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. 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
  grpcPodConfig:
    securityContextConfig: <security_mode> (3)
  image: <registry>/<namespace>/redhat-operator-index:v4.16 (4)
  displayName: My Operator Catalog
  publisher: <publisher_name> (5)
  updateStrategy:
    registryPoll: (6)
      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) カタログソースを全 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) **legacy** または **restricted** の値を指定します。フィールドが設定されていない場合、デフォルト値は **legacy** です。今後の OpenShift Container Platform リリースでは、デフォルト値が **restricted** になる予定です。**restricted** 権限でカタログを実行できない場合は、このフィールドを手動で **legacy** に設定することを推奨します。

(4) 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 (**:v4.16** 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 が常にコンテナー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。

(5) カタログを公開する名前または組織名を指定します。

(6) 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-6njx6           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

11.7. 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。

更新チャネル

Operator が複数のチャネルで利用可能な場合、サブスクライブするチャネルを選択できます。たとえば、(利用可能な場合に) **stable** チャネルからデプロイするには、これをリストから選択します。

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。

11.7.1. Web コンソールを使用した OperatorHub からのインストール

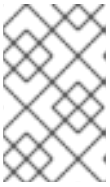
OpenShift Container Platform Web コンソールを使用して OperatorHub から Operator をインストールし、これをサブスクライブできます。

前提条件

- **cluster-admin** 権限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もしても知られる) 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。

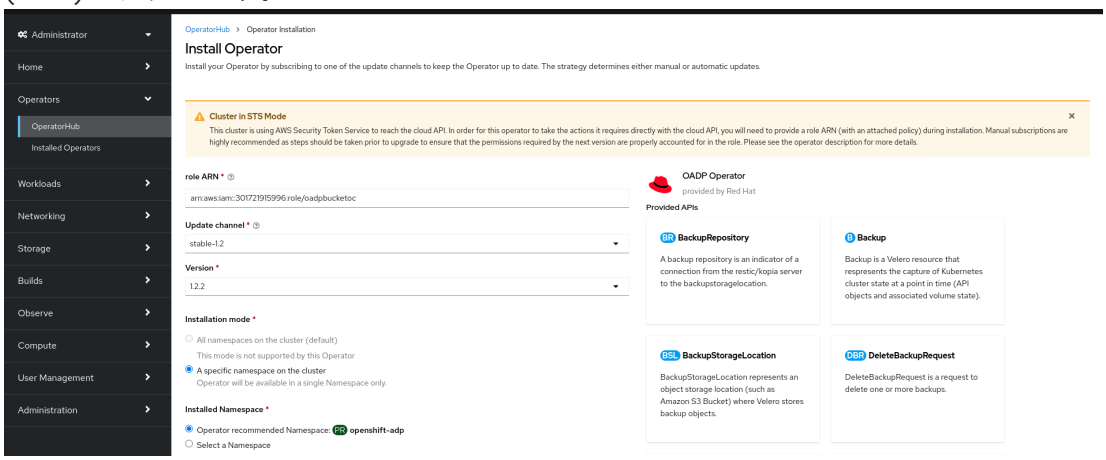


注記

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. トークン認証が有効になっているクラウドプロバイダー上のクラスターの場合:

- クラスターが AWS STS (Web コンソールの **STS モード**) を使用する場合は、**ロール ARN** フィールドにサービスアカウントの AWS IAM ロールの Amazon Resource Name (ARN) を入力します。



ロールの ARN を作成するには、AWS アカウントの準備 で説明されている手順に従います。

- クラスターが Microsoft Entra Workload ID (Web コンソールの **Workload アイデンティティー/フェデレーションされたアイデンティティーモード**) を使用する場合は、適切なフィールドにクライアント ID、テナント ID、サブスクリプション ID を追加します。

c. 複数の更新チャネルが利用可能な場合は、**Update channel** を選択します。

d. 前述のように、**Automatic** または **Manual** 承認ストラテジーを選択します。

重要

Web コンソールに、クラスターが AWS STS または Microsoft Entra Workload ID を使用していることが示されている場合は、**Update approval** を **Manual** に設定する必要があります。

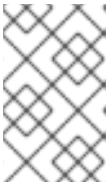
更新前に権限の変更が必要になる可能性があるため、自動更新承認のあるサブスクリプションは推奨できません。手動更新承認付きのサブスクリプションにより、管理者は新しいバージョンの権限を確認し、更新前に必要な手順を実行する機会が確保されます。

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。

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

手順

1. OperatorHub からクラスターで利用できる Operator のリストを表示します。

```
$ oc get packagemanifests -n openshift-marketplace
```

出力例

```
NAME                              CATALOG               AGE
3scale-operator                   Red Hat Operators     91m
advanced-cluster-management       Red Hat Operators     91m
amq7-cert-manager                 Red Hat Operators     91m
...
couchbase-enterprise-certified    Certified Operators   91m
crunchy-postgres-operator         Certified Operators   91m
mongodb-enterprise                Certified Operators   91m
...
etcd                              Community Operators   91m
jaeger                            Community Operators   91m
kubefed                           Community Operators   91m
...
```

必要な Operator のカタログをメモします。

2. 必要な Operator を検査して、サポートされるインストールモードおよび利用可能なチャネルを確認します。

```
$ oc describe packagemanifests <operator_name> -n openshift-marketplace
```

3. **OperatorGroup** で定義される Operator グループは、Operator グループと同じ namespace 内のすべての Operator に必要な RBAC アクセスを生成するターゲット namespace を選択します。
Operator をサブスクライブする namespace には、Operator のインストールモードに一致する Operator グループが必要になります (**AllNamespaces** または **SingleNamespace** モードのいずれか)。インストールする Operator が **AllNamespaces** モードを使用する場合、**openshift-operators** namespace には適切な **global-operators** Operator グループがすでに配置されています。

   ただし、Operator が **SingleNamespace** モードを使用し、適切な Operator グループがない場合、それらを作成する必要があります。

注記

- この手順の Web コンソールバージョンでは、**SingleNamespace** モードを選択する際に、**OperatorGroup** および **Subscription** オブジェクトの作成を背後で自動的に処理します。
- namespace ごとに Operator グループを 1 つだけ持つことができます。詳細は、「Operator グループ」を参照してください。

   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 1
spec:
  channel: <channel_name> 2
  name: <operator_name> 3
  source: redhat-operators 4
  sourceNamespace: openshift-marketplace 5
```

```
config:
  env: 6
  - name: ARGS
    value: "-v=10"
  envFrom: 7
  - secretRef:
      name: license-secret
  volumes: 8
  - name: <volume_name>
    configMap:
      name: <configmap_name>
  volumeMounts: 9
  - mountPath: <directory_name>
    name: <volume_name>
  tolerations: 10
  - operator: "Exists"
  resources: 11
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
      cpu: "500m"
  nodeSelector: 12
    foo: bar
```

1. デフォルトの **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。カスタムグローバル namespace を作成している場合はこれを指定できます。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ー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、コンテナー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ー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ー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 **NodeSelector** を定義します。

5. トークン認証が有効になっているクラウドプロバイダー上のクラスターの場合:

   a. **Subscription** オブジェクトが手動更新承認に設定されていることを確認します。

```
kind: Subscription
# ...
spec:
  installPlanApproval: Manual 1
```

1 更新前に権限の変更が必要になる可能性があるため、自動更新承認のあるサブスクリプションは推奨できません。手動更新承認付きのサブスクリプションにより、管理者は新しいバージョンの権限を確認し、更新前に必要な手順を実行する機会が確保されます。

   b. 関連するクラウドプロバイダー固有のフィールドを **Subscription** オブジェクトの **config** セクションに含めます。

   - クラスターが AWS STS モードの場合は、次のフィールドを含めます。

```
kind: Subscription
# ...
spec:
  config:
    env:
    - name: ROLEARN
      value: "<role_arn>" 1
```

1 ロール ARN の詳細を含めます。

   - クラスターが Microsoft Entra Workload ID モードの場合は、次のフィールドを含めます。

```
kind: Subscription
# ...
spec:
 config:
   env:
   - name: CLIENTID
     value: "<client_id>" 1
   - name: TENANTID
     value: "<tenant_id>" 2
   - name: SUBSCRIPTIONID
     value: "<subscription_id>" 3
```

1 クライアント ID を含めます。

2 テナント ID を含めます。

3 サブスクリプション ID を含めます。

6. **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12章 クラウドプロバイダーの認証情報の設定変更

サポートされている構成では、OpenShift Container Platform がクラウドプロバイダーに対して認証する方法を変更できます。

クラスターが使用するクラウド認証情報戦略を確認するには、Cloud Credential Operator モードの決定を参照してください。

12.1. 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。

12.1.1. 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認証情報のローテーション

Cloud Credential Operator (CCO) ユーティリティー **ccoctl** 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。

12.1.1.1. 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

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

手順

- **ccoctl** 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。

```
$ ccoctl <provider_name> refresh-keys \ 1
    --kubeconfig <openshift_kubeconfig_file> \ 2
    --credentials-requests-dir <path_to_credential_requests_directory> \ 3
    --name <name> 4
```

1. プロバイダーの名前。例: **ibmcloud** または **powervs**。
2. 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。
3. 認証情報の要求が保存されるディレクトリー。
4. 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。

12.1.2. クラウドプロバイダーの認証情報の維持

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 を使用するように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 モードの場合、Amazon Web Services (AWS)、Microsoft Azure、Google Cloud Platform (GCP)、Red Hat OpenStack Platform (RHOSP)、および VMware vSphere がサポートされます。
- 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

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**
Azure	**azure-credentials**
GCP	**gcp-credentials**
RHOSP	**openstack-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 コントローラー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
```

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> \ 1
  -n <secret_namespace> 2
```

1 シークレットの名前を指定します。

2 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

12.1.3. 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

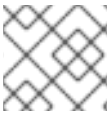
- 管理者の認証情報のルートシークレット形式

12.2. トークンベースの認証の有効化

Microsoft Azure OpenShift Container Platform クラスターをインストールした後、Microsoft Entra Workload ID を有効にして短期認証情報を使用できます。

12.2.1. Cloud Credential Operator ユーティリティーの設定

Cloud Credential Operator (CCO) が手動モードで動作しているときにクラスターの外部からクラウドクレデンシャルを作成および管理するには、CCO ユーティリティー (**ccoctl**) バイナリーを抽出して準備します。



注記

ccoctl ユーティリティーは、Linux 環境で実行する必要がある Linux バイナリーです。

前提条件

- クラスター管理者のアクセスを持つ OpenShift Container Platform アカウントを使用できる。
- OpenShift CLI (**oc**) がインストールされている。

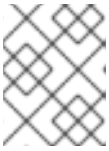
手順

1. 次のコマンドを実行して、OpenShift Container Platform リリースイメージの変数を設定します。

```
$ RELEASE_IMAGE=$(./openshift-install version | awk '/release image/ {print $3}')
```

2. 以下のコマンドを実行して、OpenShift Container Platform リリースイメージから CCO コンテナーイメージを取得します。

```
$ CCO_IMAGE=$(oc adm release info --image-for='cloud-credential-operator'
$RELEASE_IMAGE -a ~/.pull-secret)
```



注記

$RELEASE_IMAGE のアーキテクチャーが、**ccoctl**ツールを使用する環境のアーキテクチャーと一致していることを確認してください。

3. 以下のコマンドを実行して、OpenShift Container Platform リリースイメージ内の CCO コンテナーイメージから **ccoctl** バイナリーを抽出します。

```
$ oc image extract $CCO_IMAGE \
  --file="/usr/bin/ccoctl.<rhel_version>" \ 1
  -a ~/.pull-secret
```

1 **<rhel_version>** には、ホストが使用する Red Hat Enterprise Linux (RHEL) のバージョンに対応する値を指定します。値が指定されていない場合は、デフォルトで **ccoctl.rhel8** が使用されます。次の値が有効です。

- **rhel8**: RHEL 8 を使用するホストの場合はこの値を指定します。
- **rhel9**: RHEL 9 を使用するホストの場合はこの値を指定します。

4. 次のコマンドを実行して、権限を変更して **ccoctl** を実行可能にします。

```
$ chmod 775 ccoctl.<rhel_version>
```

検証

- **ccoctl** を使用する準備ができていることを確認するには、次のコマンドを実行してヘルプファイルを表示します。

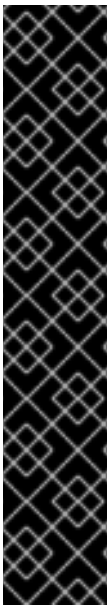
```
$ ccoctl --help
```

ccoctl --help の出力

```
OpenShift credentials provisioning tool

Usage:
  ccoctl [command]

Available Commands:
  aws         Manage credentials objects for AWS cloud
  azure       Manage credentials objects for Azure
  gcp         Manage credentials objects for Google cloud
  help        Help about any command
  ibmcloud    Manage credentials objects for IBM Cloud
  nutanix     Manage credentials objects for Nutanix

Flags:
  -h, --help   help for ccoctl

Use "ccoctl [command] --help" for more information about a command.
```

12.2.2. 既存のクラスターで Microsoft Entra Workload ID を有効にする

インストール中に Microsoft Azure OpenShift Container Platform クラスターを Microsoft Entra Workload ID を使用するように設定しなかった場合は、既存のクラスターでこの認証方法を有効にすることができます。



重要

既存のクラスターで Workload ID を有効にするプロセスは、サービスの停止を伴い、かなりの時間がかかります。続行する前に、次の考慮事項を確認してください。

- 次の手順を読み、必ず時間の要件を理解してご承知おきください。正確な所要時間は個々のクラスターによって異なりますが、少なくとも 1 時間かかる可能性があります。
- このプロセス中に、すべてのサービスアカウントを更新し、クラスター上のすべての Pod を再起動する必要があります。これらのアクションはワークロードの停止を伴います。この影響を軽減するには、これらのサービスを一時的に停止し、クラスターの準備ができたときに再デプロイすることができます。
- このプロセスを開始した後は、完了するまでクラスターの更新を試みないでください。更新がトリガーされると、既存のクラスターで Workload ID を有効にするプロセスが失敗します。

前提条件

- Microsoft Azure に OpenShift Container Platform クラスターをインストールした。
- **cluster-admin** 権限を持つアカウントを使用してクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- Cloud Credential Operator ユーティリティー (**ccoctl**) バイナリーを展開して準備した。
- Azure CLI (**az**) を使用して Azure アカウントにアクセスできる。

手順

1. **ccoctl** ユーティリティーが生成するマニフェストの出力ディレクトリーを作成します。この手順では、**./output_dir** を例として使用します。
2. 次のコマンドを実行して、クラスターのサービスアカウントの公開署名鍵を出力ディレクトリーに抽出します。

```
$ oc get configmap \
  --namespace openshift-kube-apiserver bound-sa-token-signing-certs \
  --output 'go-template={{index .data "service-account-001.pub"}}' >
./output_dir/serviceaccount-signer.public 1
```

1 この手順では、例として **serviceaccount-signer.public** という名前のファイルを使用します。

3. 抽出したサービスアカウント公開署名鍵を使用して、次のコマンドを実行し、OpenID Connect (OIDC) 発行者と、OIDC 設定ファイルを含む Azure Blob ストレージコンテナーを作成します。

```
$ ./ccoctl azure create-oidc-issuer \
  --name <azure_infra_name> \ 1
  --output-dir ./output_dir \
  --region <azure_region> \ 2
  --subscription-id <azure_subscription_id> \ 3
  --tenant-id <azure_tenant_id> \
  --public-key-file ./output_dir/serviceaccount-signer.public 4
```

1. **name** パラメーターの値は、Azure リソースグループを作成するために使用します。新しい Azure リソースグループを作成する代わりに既存の Azure リソースグループを使用するには、**--oidc-resource-group-name** を指定し、その値として既存のグループ名を使用します。
2. 既存のクラスターのリージョンを指定します。
3. 既存のクラスターのサブスクリプション ID を指定します。
4. クラスターのサービスアカウントの公開署名鍵を含むファイルを指定します。

4. 次のコマンドを実行して、Azure pod identity webhook の設定ファイルが作成されたことを確認します。

```
$ ll ./output_dir/manifests
```

出力例

```
total 8
-rw-------. 1 cloud-user cloud-user 193 May 22 02:29 azure-ad-pod-identity-webhook-
config.yaml 1
-rw-------. 1 cloud-user cloud-user 165 May 22 02:29 cluster-authentication-02-config.yaml
```

1. ファイル **azure-ad-pod-identity-webhook-config.yaml** には、Azure pod identity webhook 設定が含まれています。

5. 次のコマンドを実行して、出力ディレクトリーに生成されたマニフェストの OIDC 発行者 URL を使用して **OIDC_ISSUER_URL** 変数を設定します。

```
$ OIDC_ISSUER_URL=`awk '/serviceAccountIssuer/ { print $2 }'
./output_dir/manifests/cluster-authentication-02-config.yaml`
```

6. 次のコマンドを実行して、クラスターの **authentication** 設定の **spec.serviceAccountIssuer** パラメーターを更新します。

```
$ oc patch authentication cluster \
  --type=merge \
  -p "{\"spec\":{\"serviceAccountIssuer\":\"${OIDC_ISSUER_URL}\"}}"
```

7. 次のコマンドを実行して、設定の更新の進行状況を監視します。

```
$ oc adm wait-for-stable-cluster
```

このプロセスには 15 分以上かかる場合があります。次の出力は、プロセスが完了したことを示します。

```
All clusteroperators are stable
```

8. 次のコマンドを実行して、クラスター内のすべての Pod を再起動します。

```
$ oc adm reboot-machine-config-pool mcp/worker mcp/master
```

Pod を再起動すると、**serviceAccountIssuer** フィールドが更新され、サービスアカウントの公開署名鍵が更新されます。

9. 次のコマンドを実行して、再起動と更新のプロセスを監視します。

```
$ oc adm wait-for-node-reboot nodes --all
```

このプロセスには 15 分以上かかる場合があります。次の出力は、プロセスが完了したことを示します。

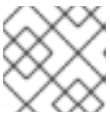
```
All nodes rebooted
```

10. 次のコマンドを実行して、Cloud Credential Operator の **spec.credentialsMode** パラメーターを **Manual** に更新します。

```
$ oc patch cloudcredential cluster \
  --type=merge \
  --patch '{"spec":{"credentialsMode":"Manual"}}'
```

11. 以下のコマンドを実行して、OpenShift Container Platform リリースイメージから **CredentialsRequest** オブジェクトのリストを抽出します。

```
$ oc adm release extract \
  --credentials-requests \
  --included \
  --to <path_to_directory_for_credentials_requests> \
  --registry-config ~/.pull-secret
```



注記

このコマンドの実行には少し時間がかかる場合があります。

12. 次のコマンドを実行して、Azure リソースグループ名を使用して **AZURE_INSTALL_RG** 変数を設定します。

```
$ AZURE_INSTALL_RG=`oc get infrastructure cluster -o jsonpath --template '{
.status.platformStatus.azure.resourceGroupName }'`
```

13. 次のコマンドを実行して、**ccoctl** ユーティリティーを使用してすべての **CredentialsRequest** オブジェクトのマネージド ID を作成します。

```
$ ccoctl azure create-managed-identities \
  --name <azure_infra_name> \
```

```
--output-dir ./output_dir \
--region <azure_region> \
--subscription-id <azure_subscription_id> \
--credentials-requests-dir <path_to_directory_for_credentials_requests> \
--issuer-url "${OIDC_ISSUER_URL}" \
--dnszone-resource-group-name <azure_dns_zone_resourcegroup_name> \ 1
--installation-resource-group-name "${AZURE_INSTALL_RG}"
```

1 DNS ゾーンを含むリソースグループの名前を指定します。

14. 次のコマンドを実行して、Workload ID の Azure pod identity webhook 設定を適用します。

```
$ oc apply -f ./output_dir/manifests/azure-ad-pod-identity-webhook-config.yaml
```

15. 次のコマンドを実行して、**ccoctl** ユーティリティーによって生成されたシークレットを適用します。

```
$ find ./output_dir/manifests -iname "openshift*yaml" -print0 | xargs -I {} -0 -t oc replace -f {}
```

このプロセスには数分の時間がかかる可能性があります。

16. 次のコマンドを実行して、クラスター内のすべての Pod を再起動します。

```
$ oc adm reboot-machine-config-pool mcp/worker mcp/master
```

Pod を再起動すると、**serviceAccountIssuer** フィールドが更新され、サービスアカウントの公開署名鍵が更新されます。

17. 次のコマンドを実行して、再起動と更新のプロセスを監視します。

```
$ oc adm wait-for-node-reboot nodes --all
```

このプロセスには 15 分以上かかる場合があります。次の出力は、プロセスが完了したことを示します。

```
All nodes rebooted
```

18. 次のコマンドを実行して、設定の更新の進行状況を監視します。

```
$ oc adm wait-for-stable-cluster
```

このプロセスには 15 分以上かかる場合があります。次の出力は、プロセスが完了したことを示します。

```
All clusteroperators are stable
```

19. オプション: 次のコマンドを実行して、Azure の root 認証情報シークレットを削除します。

```
$ oc delete secret -n kube-system azure-credentials
```

関連情報

- Microsoft Entra Workload ID
- 短期認証情報を使用するように Azure クラスターを設定する

12.2.3. クラスターが短期認証情報を使用していることを確認する

クラスター内の Cloud Credential Operator (CCO) 設定やその他の値を確認することで、クラスターが個々のコンポーネントに対して短期的なセキュリティー認証情報を使用していることを確認できます。

前提条件

- Cloud Credential Operator ユーティリティー (**ccoctl**) を使用して OpenShift Container Platform クラスターをデプロイし、短期認証情報を実装しました。
- 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

- 次のコマンドを実行して、CCO が手動モードで動作するように設定されていることを確認します。

```
$ oc get cloudcredentials cluster \
  -o=jsonpath={.spec.credentialsMode}
```

次の出力は、CCO が手動モードで動作していることを示しています。

出力例

```
Manual
```

- 次のコマンドを実行して、クラスターに **root** 認証情報がないことを確認します。

```
$ oc get secrets \
  -n kube-system <secret_name>
```

<secret_name> は、クラウドプロバイダーのルートシークレットの名前です。

プラットフォーム	シークレット名
Amazon Web Services (AWS)	**aws-creds**
Microsoft Azure	**azure-credentials**
Google Cloud Platform (GCP)	**gcp-credentials**

エラーは、ルートシークレットがクラスター上に存在しないことを確認します。

AWS クラスターの出力例

```
Error from server (NotFound): secrets "aws-creds" not found
```

- 次のコマンドを実行して、コンポーネントが個々のコンポーネントに対して短期セキュリティー認証情報を使用していることを確認します。

```
$ oc get authentication cluster \
  -o jsonpath \
  --template='{ .spec.serviceAccountIssuer }'
```

このコマンドは、クラスター **Authentication** オブジェクトの **.spec.serviceAccountIssuer** パラメーターの値を表示します。クラウドプロバイダーに関連付けられた URL の出力は、クラスターがクラスターの外部から作成および管理される短期認証情報を使用して手動モードを使用していることを示します。

- Azure クラスター: 次のコマンドを実行して、コンポーネントがシークレットマニフェストで指定された Azure クライアント ID を想定していることを確認します。

```
$ oc get secrets \
  -n openshift-image-registry installer-cloud-credentials \
  -o jsonpath='{.data}'
```

出力に **azure_client_id** フィールドと **azure_federated_token_file** フィールドが含まれていれば、コンポーネントが Azure クライアント ID を想定しています。

- Azure クラスター: 次のコマンドを実行して、pod identity webhook が実行されていることを確認します。

```
$ oc get pods \
  -n openshift-cloud-credential-operator
```

出力例

```
NAME                                        READY   STATUS    RESTARTS   AGE
cloud-credential-operator-59cf744f78-r8pbq  2/2     Running   2          71m
pod-identity-webhook-548f977b4c-859lz       1/1     Running   1          70m
```

12.3. 関連情報

- Cloud Credential Operator について

第13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。

13.1. 外部システムへの通知の送信

OpenShift Container Platform 4.16 では、実行するアラートをアラート UI に表示できます。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
- Microsoft Teams

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。

13.2. 関連情報

- モニタリングの概要
- アラートレシーバーの設定

第14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。

14.1. 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 などのコンテナーミラーレジストリーにミラーリングできます。大規模なコンテナー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ー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 リポジトリー、Harbor など、Dockerv2-2 をサポートする任意のコンテナー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。

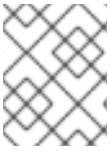
重要

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 以外のコンテナー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。

14.2. 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ー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 リポジトリー
  - 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の 概念実証の目的、または Quay Operator の使用 を参照してください。
- ミラーリポジトリーは、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 リポジトリーでは、イメージを共有するために Organizations が必要です。
- 必要なコンテナー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。

14.3. 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 のリストに追加する。
- **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 のリストに追加します。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、以下のようになります。, </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 の場合は以下のようになります。

```
$ 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、以下のようになります。

<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":"b3BlbnNoaWZ0Y3UjhGOVZPT0lOMEFaUjdPUzRGTA==","email":"user@exa
mple.com"},
"quay.io":
{"auth":"b3BlbnNoaWZ0LXJlbGVhc2UtZGOVZPT0lOMEFaUGSTd4VGVGVUjdPUzR
GTA==","email":"user@example.com"},
"registry.connect.redhat.com"
{"auth":"NTE3MTMwNDB8dWhjLTFEZlN3VHkxOSTd4VGVGVU1MdTpleUpoYkdjaUail
A==","email":"user@example.com"},
"registry.redhat.io":
{"auth":"NTE3MTMwNDB8dWhjLTFEZlN3VH3BGSTd4VGVGVU1MdTpleUpoYkdjaU9
fZw==","email":"user@example.com"},
"registry.svc.ci.openshift.org":
{"auth":"dXNlcjpyWjAwWVFjSEJiT2RKVW1pSmg4dW92dGp1SXRxQ3RGN1pwajJhN1
ZXeTRV"},"my-registry:5000/my-namespace/":
{"auth":"dXNlcm5hbWU6cGFzc3dvcmQ="}}}
```

14.4. 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、外部リポジトリーからミラーリポジトリー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**)。

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、以下のようになります。

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。

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ー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

ミラーリポジトリーのレジストリー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ー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。

14.5. 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。

```
S 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ー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 (1)
    source: quay.io/openshift-release-dev/ocp-release (2)
  - mirrors:
    - mirror.registry.com:443/ocp/release
    source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

   (1) ミラーイメージレジストリーおよびリポジトリーの名前を指定します。

   (2) 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ーを指定します。

   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
```

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
```

1 ミラーレジストリーがリスト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
```

14.6. 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                       NAME                                                READY
STATUS     RESTARTS   AGE
kube-system                     apiserver-watcher-ci-ln-47ltxtb-f76d1-mrffg-master-0
1/1     Running    0          39m
kube-system                     apiserver-watcher-ci-ln-47ltxtb-f76d1-mrffg-master-1
1/1     Running    0          39m
kube-system                     apiserver-watcher-ci-ln-47ltxtb-f76d1-mrffg-master-2
1/1     Running    0          39m
openshift-apiserver-operator          openshift-apiserver-operator-79c7c646fd-5rvr5
1/1     Running    3          45m
openshift-apiserver             apiserver-b944c4645-q694g                           2/2
Running    0          29m
openshift-apiserver             apiserver-b944c4645-shdxb                           2/2
Running    0          31m
openshift-apiserver             apiserver-b944c4645-x7rf2                           2/2
Running    0          33m
 ...
```

- 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-47ltxtb-f76d1-mrffg-master-0       Ready    master   42m   v1.29.4
ci-ln-47ltxtb-f76d1-mrffg-master-1       Ready    master   42m   v1.29.4
ci-ln-47ltxtb-f76d1-mrffg-master-2       Ready    master   42m   v1.29.4
ci-ln-47ltxtb-f76d1-mrffg-worker-a-gsxbz   Ready    worker   35m   v1.29.4
ci-ln-47ltxtb-f76d1-mrffg-worker-b-5qqdx   Ready    worker   35m   v1.29.4
ci-ln-47ltxtb-f76d1-mrffg-worker-c-rjkpq   Ready    worker   34m   v1.29.4
```

14.7. 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ーをミラーリングし、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。

14.8. 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
```

14.9. 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"] 1
   ```

   1 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15章 クラスター機能の有効化

クラスター管理者は、インストール前に無効化されたクラスター機能を有効化できます。

注記

クラスター管理者は、クラスター機能を有効にした後、それを無効にすることはできません。

15.1. クラスター機能の表示

クラスター管理者は、**clusterversion** リソースの状態を使用して機能を表示できます。

前提条件

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

手順

- クラスター機能の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 clusterversion version -o jsonpath='{.spec.capabilities}{"\n"}{.status.capabilities}{"\n"}'
```

出力例

```
{"additionalEnabledCapabilities":["openshift-samples"],"baselineCapabilitySet":"None"}
{"enabledCapabilities":["openshift-samples"],"knownCapabilities":
["CSISnapshot","Console","Insights","Storage","baremetal","marketplace","openshift-
samples"]}
```

15.2. クラスター機能を有効にするベースライン機能セットの設定

クラスター管理者は、**baselineCapabilitySet** を設定して機能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

- **baselineCapabilitySet** を設定するには、次のコマンドを実行します。

```
$ oc patch clusterversion version --type merge -p '{"spec":{"capabilities":
{"baselineCapabilitySet":"vCurrent"}}}' (1)
```

(1) **baselineCapabilitySet** には、**vCurrent**、**v4.16**、または **None** を指定できます。

次の表では、**baselineCapabilitySet** の値について説明します。

表15.1 クラスター機能の baselineCapabilitySet 値の説明

値	説明
vCurrent	新しいリリースで導入される新しいデフォルト機能を自動的に追加する場合、このオプションを指定します。
v4.11	OpenShift Container Platform 4.11 のデフォルト機能を有効にする場合、このオプションを指定します。**v4.11** を指定すると、それ以降のバージョンの OpenShift Container Platform で導入された機能が有効になりません。OpenShift Container Platform 4.11 のデフォルト機能は、**baremetal**、**MachineAPI**、**marketplace**、および **openshift-samples** です。
v4.12	OpenShift Container Platform 4.12 のデフォルト機能を有効にする場合、このオプションを指定します。**v4.12** を指定すると、それ以降のバージョンの OpenShift Container Platform で導入された機能が有効になりません。OpenShift Container Platform 4.12 のデフォルト機能は、**baremetal**、**MachineAPI**、**marketplace**、**openshift-samples**、**Console**、**Insights**、**Storage**、および **CSISnapshot** です。
v4.13	OpenShift Container Platform 4.13 のデフォルト機能を有効にする場合、このオプションを指定します。**v4.13** を指定すると、それ以降のバージョンの OpenShift Container Platform で導入された機能が有効になりません。OpenShift Container Platform 4.13 のデフォルト機能は、**baremetal**、**MachineAPI**、**marketplace**、**openshift-samples**、**Console**、**Insights**、**Storage**、**CSISnapshot**、および **NodeTuning** です。
v4.14	OpenShift Container Platform 4.14 のデフォルト機能を有効にする場合、このオプションを指定します。**v4.14** を指定すると、それ以降のバージョンの OpenShift Container Platform で導入された機能が有効になりません。OpenShift Container Platform 4.14 のデフォルト機能は、**baremetal**、**MachineAPI**、**marketplace**、**openshift-samples**、**Console**、**Insights**、**Storage**、**CSISnapshot**、**NodeTuning**、**ImageRegistry**、**Build**、および **DeploymentConfig** です。
v4.15	OpenShift Container Platform 4.15 のデフォルト機能を有効にする場合、このオプションを指定します。**v4.15** を指定すると、それ以降のバージョンの OpenShift Container Platform で導入された機能が有効になりません。OpenShift Container Platform 4.15 のデフォルト機能は、**baremetal**、**MachineAPI**、**marketplace**、**OperatorLifecycleManager**、**openshift-samples**、**Console**、**Insights**、**Storage**、**CSISnapshot**、**NodeTuning**、**ImageRegistry**、**Build**、**CloudCredential**、および **DeploymentConfig** です。

値	説明
v4.16	OpenShift Container Platform 4.16 のデフォルト機能を有効にする場合、このオプションを指定します。**v4.16** を指定すると、それ以降のバージョンの OpenShift Container Platform で導入された機能が有効になりません。OpenShift Container Platform 4.16 のデフォルトの機能は、**baremetal**、**MachineAPI**、**marketplace**、**OperatorLifecycleManager**、**openshift-samples**、**Console**、**Insights**、**Storage**、**CSISnapshot**、**NodeTuning**、**ImageRegistry**、**Build**、**CloudCredential**、**DeploymentConfig**、および **CloudControllerManager** です。
None	他のセットが大きすぎる場合や、機能が必要ない場合、**additionalEnabledCapabilities** を介して微調整する場合に指定します。

15.3. 追加で有効な機能を設定することによるクラスター機能の有効化

クラスター管理者は、**additionalEnabledCapabilities** を設定してクラスター機能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

1. 次のコマンドを実行して、追加の有効な機能を表示します。

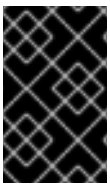
```
$ oc get clusterversion version -o jsonpath='{.spec.capabilities.additionalEnabledCapabilities}{"\n"}'
```

出力例

```
["openshift-samples"]
```

2. **additionalEnabledCapabilities** を設定するには、次のコマンドを実行します。

```
$ oc patch clusterversion/version --type merge -p '{"spec":{"capabilities":{"additionalEnabledCapabilities":["openshift-samples", "marketplace"]}}}'
```



重要

クラスターですでに有効になっている機能を無効にすることはできません。クラスターバージョン Operator (CVO) は、クラスターですでに有効になっている機能を調整し続けます。

機能を無効にしようとすると、CVO は異なる仕様を示します。

```
$ oc get clusterversion version -o jsonpath='{.status.conditions[?
(@.type=="ImplicitlyEnabledCapabilities")]}{"\n"}'
```

出力例

```
{"lastTransitionTime":"2022-07-22T03:14:35Z","message":"The following capabilities could not be
disabled: openshift-
samples","reason":"CapabilitiesImplicitlyEnabled","status":"True","type":"ImplicitlyEnabledCapabilities"}
```

注記

クラスターのアップグレード中に、特定の機能が暗黙的に有効になる可能性があります。アップグレード前にクラスターでリソースがすでに実行されていた場合には、そのリソースに含まれるすべての機能が有効になります。たとえば、クラスターのアップグレード中に、そのクラスターですでに実行中のリソースが、システムにより **marketplace** 機能に含まれるように、変更される場合などです。クラスター管理者が **marketplace** 機能を明示的に有効にしていなくても、システムによって暗黙的に有効にされています。

15.4. 関連情報

- クラスター機能

第16章 IBM Z または IBM LINUXONE 環境での追加デバイスの設定

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 IBM® 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 IBM® 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

- マシン設定の概要

16.1. 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 IBM® 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: ""
```

16.1.1. 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 (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-host-0.0.8000.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。

16.1.2. 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。

16.1.3. 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。

16.1.4. 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** の例を見てみましょう。

```
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

16.2. 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 IBM® 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

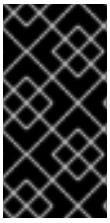
IBM® ドキュメントの Persistent device configuration を参照してください。

16.3. 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。

16.4. 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 IBM® 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 IBM® 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 IBM® 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

第17章 VMWARE VSPHERE 上のクラスターの複数のリージョンとゾーンの設定

管理者は、VMware vSphere インスタンス上で実行される OpenShift Container Platform クラスターに複数のリージョンとゾーンを指定できます。この設定により、ハードウェアの障害やネットワークの停止によってクラスターに障害が発生するリスクが軽減されます。

障害ドメイン設定には、トポロジーを作成するパラメーターがリストされます。次のリストは、これらのパラメーターの一部を示しています。

- **computeCluster**
- **datacenter**
- **datastore**
- **networks**
- **resourcePool**

OpenShift Container Platform クラスターに複数のリージョンとゾーンを定義した後、ノードを作成したり、別の障害ドメインにノードを移行したりできます。

重要

既存の OpenShift Container Platform クラスターコンピュートノードを障害ドメインに移行する場合は、コンピュートノード用に新しいコンピューティングマシンセットを定義する必要があります。この新しいマシンセットは、障害ドメインのトポロジーに応じてコンピュートノードをスケールアップし、既存のコンピュートノードをスケールダウンできます。

クラウドプロバイダーは、マシンセットリソースによってプロビジョニングされたコンピュートノードに、**topology.kubernetes.io/zone** ラベルと **topology.kubernetes.io/region** ラベルを追加します。

詳細は、コンピュートマシンセットの作成 を 参照してください。

17.1. VSPHERE 上のクラスターに複数のリージョンとゾーンを指定する

Infrastructures.config.openshift.io 設定リソースを設定して、VMware vSphere インスタンス上で実行される OpenShift Container Platform クラスターに複数のリージョンとゾーンを指定できます。

クラウドコントローラーマネージャーおよび vSphere Container Storage Interface (CSI) Operator Driver のトポロジー認識機能には、OpenShift Container Platform クラスターをホストする vSphere トポロジーに関する情報が必要です。このトポロジー情報は、**infrastructors.config.openshift.io** 設定リソースに存在します。

クラスターのリージョンとゾーンを指定する前に、クラウドプロバイダーがノードにラベルを追加できるように、すべてのデータセンターとコンピューティングクラスターにタグが含まれていることを確認する必要があります。たとえば、**datacenter-1** が **リージョン a** を表し、**compute-cluster-1** が **zone-1** を表す場合、クラウドプロバイダーは、**region-a** の値を持つ **openshift-region** カテゴリーラベルを **datacenter-1** に追加します。さらに、クラウドプロバイダーは、**zone-1** の値を持つ **openshift-zone** カテゴリータグを **compute-cluster-1** に追加します。

注記

vMotion 機能を備えたコントロールプレーンノードを障害ドメインに移行できます。これらのノードを障害ドメインに追加すると、クラウドプロバイダーはこれらのノードに **topology.kubernetes.io/zone** ラベルと **topology.kubernetes.io/region** ラベルを追加します。

前提条件

- vCenter サーバー上に **openshift-region** タグカテゴリーと **openshift-zone** タグカテゴリーを作成しました。
- 各データセンターとコンピューティングクラスターに、関連付けられたリージョンまたはゾーン、あるいはその両方の名前を表すタグが含まれていることを確認しました。
- オプション: **API** および **Ingress** 静的 IP アドレスをインストールプログラムに定義した場合は、すべてのリージョンとゾーンが共通のレイヤー 2 ネットワークを共有していることを確認する必要があります。この設定により、API および Ingress Virtual IP (VIP) アドレスがクラスターと対話できるようになります。

重要

ノードを作成するかノードを移行する前に、すべてのデータセンターとコンピューティングクラスターにタグを提供しない場合、クラウドプロバイダーは、**topology.kubernetes.io/zone** ラベルと **topology.kubernetes.io/region** ラベルをノードに追加できません。これは、サービスがトラフィックをノードにルーティングできないことを意味します。

手順

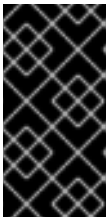
1. 次のコマンドを実行して、クラスターの **infrastructures.config.openshift.io** カスタムリソース定義 (CRD) を編集して、リソースの **failureDomains** セクションに複数のリージョンとゾーンを指定します。

```
$ oc edit infrastructures.config.openshift.io cluster
```

複数のリージョンおよびゾーンが設定で定義された cluster という名前のインスタンスの infrastructures.config.openshift.io CRD の例

```
spec:
  cloudConfig:
    key: config
    name: cloud-provider-config
  platformSpec:
    type: vSphere
    vsphere:
      vcenters:
        - datacenters:
            - <region_a_datacenter>
            - <region_b_datacenter>
          port: 443
          server: <your_vcenter_server>
      failureDomains:
        - name: <failure_domain_1>
          region: <region_a>
```

```
      zone: <zone_a>
      server: <your_vcenter_server>
      topology:
        datacenter: <region_a_dc>
        computeCluster: "</region_a_dc/host/zone_a_cluster>"
        resourcePool: "</region_a_dc/host/zone_a_cluster/Resources/resource_pool>"
        datastore: "</region_a_dc/datastore/datastore_a>"
        networks:
        - port-group
    - name: <failure_domain_2>
      region: <region_a>
      zone: <zone_b>
      server: <your_vcenter_server>
      topology:
        computeCluster: </region_a_dc/host/zone_b_cluster>
        datacenter: <region_a_dc>
        datastore: </region_a_dc/datastore/datastore_a>
        networks:
        - port-group
    - name: <failure_domain_3>
      region: <region_b>
      zone: <zone_a>
      server: <your_vcenter_server>
      topology:
        computeCluster: </region_b_dc/host/zone_a_cluster>
        datacenter: <region_b_dc>
        datastore: </region_b_dc/datastore/datastore_b>
        networks:
        - port-group
  nodeNetworking:
    external: {}
    internal: {}
```



重要

障害ドメインを作成し、それを VMware vSphere クラスターの CRD で定義した後は、障害ドメインを変更または削除しないでください。この設定でこれらのアクションのいずれかを実行すると、コントロールプレーンマシンの可用性とフォールトトレランスに影響を与える可能性があります。

2. リソースファイルを保存して変更を適用します。

関連情報

- クラスター全体のインフラストラクチャー CRD のパラメーター

17.2. クラスターで複数のレイヤー 2 ネットワークを有効にする

ノード間のデータ転送が複数のネットワークにまたがるように、複数のレイヤー 2 ネットワーク設定を使用するようにクラスターを設定できます。

前提条件

- クラスターコンポーネントが相互に通信できるように、マシン間のネットワーク接続を設定しました。

手順

- installer-provisioned infrastructure を使用してクラスターをインストールした場合は、すべてのコントロールプレーンノードが共通のレイヤー 2 ネットワークを共有していることを確認する必要があります。さらに、Ingress Pod スケジューリング用に設定されたコンピュートノードが共通のレイヤー 2 ネットワークを共有していることを確認します。
  - 複数のレイヤー 2 ネットワークにまたがるコンピュートノードが必要な場合は、Ingress Pod をホストできるインフラストラクチャーノードを作成できます。
  - 追加のレイヤー 2 ネットワークにわたってワークロードをプロビジョニングする必要がある場合は、vSphere 上にコンピューティングマシンセットを作成し、これらのワークロードをターゲットのレイヤー 2 ネットワークに移動できます。
- ユーザーが提供したインフラストラクチャー (user-provisioned infrastructure として定義) にクラスターをインストールした場合は、ニーズを満たすために次のアクションを実行します。
  - ロードバランサーがコントロールプレーンノード上の API およびマシン設定サーバーにアクセスできるように、API ロードバランサーとネットワークを設定します。
  - ロードバランサーがコンピュートノードまたはインフラストラクチャーノード上の Ingress Pod に到達できるように、Ingress ロードバランサーとネットワークを設定します。

関連情報

- ネットワーク接続の要件
- 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
- コンピュートマシンセットの作成

17.3. クラスター全体のインフラストラクチャー CRD のパラメーター

VMware vSphere インスタンス上で実行される OpenShift Container Platform クラスターの複数のリージョンとゾーンを定義するには、クラスター全体のインフラストラクチャー、**infrastructs.config.openshift.io**、カスタムリソース定義 (CRD) の特定のパラメーターの値を設定する必要があります。

次の表に、OpenShift Container Platform クラスターの複数のリージョンとゾーンを定義するための必須パラメーターを示します。

パラメーター	説明
vcenters	OpenShift Container Platform クラスターの vCenter サーバー。クラスターに指定できる vCenter は 1 つだけです。
datacenters	OpenShift Container Platform クラスターに関連付けられた VM が作成されるか、現在存在する vCenter データセンター。
port	vCenter サーバーの TCP ポート。

パラメーター	説明
server	vCenter サーバーの完全修飾ドメイン名 (FQDN)。
failureDomains	障害が発生したドメインのリスト。
name	障害ドメインの名前。
region	障害障害ドメインのトポロジーに割り当てられた **openshift-region** タグの値。
zone	障害障害ドメインのトポロジーに割り当てられた **openshift-zone** タグの値。
topology	障害ドメインに関連付けられた vCenter リソース。
datacenter	障害ドメインに関連付けられたデータセンター。
computeCluster	障害ドメインに関連付けられたコンピューティングクラスターのフルパス。
resourcePool	障害ドメインに関連付けられたリソースプールのフルパス。
datastore	障害ドメインに関連付けられたデータストアのフルパス。
networks	障害ドメインに関連付けられたポートグループのリスト。定義できる portgroup は 1 つだけです。

関連情報

- vSphere 上のクラスターに複数のリージョンとゾーンを指定する

第18章 既存の NUTANIX クラスターへの障害ドメインの追加

デフォルトでは、インストールプログラムは、コントロールプレーンとコンピュートマシンを単一の Nutanix Prism Element (クラスター) にインストールします。OpenShift Container Platform クラスターをデプロイした後、障害ドメインを使用して追加の Prism Element インスタンスをデプロイメントに追加することで、フォールトトレランスを向上させることができます。

障害ドメインは、単一の Prism Element インスタンスを表します。障害ドメインに対しては、以下を実行できます。

- 新しいコントロールプレーンとコンピュートマシンをデプロイする
- 既存のコントロールプレーンとコンピュートマシンを分散する

18.1. 障害ドメインの要件

障害ドメインの使用を計画する場合は、次の要件を考慮してください。

- すべての Nutanix Prism Element インスタンスは、同一の Prism Central インスタンスによって管理する必要があります。複数の Prism Central インスタンスで構成されるデプロイメントはサポートされていません。
- Prism Element クラスターを構成するマシンは、障害ドメインが相互に通信できるように、同じイーサネットネットワーク上に存在する必要があります。
- OpenShift Container Platform クラスターで障害ドメインとして使用する各 Prism Element には、サブネットが必要です。これらのサブネットを定義するときは、共通の IP アドレス接頭辞 (CIDR) を指定する必要があります。また、OpenShift Container Platform クラスターが使用する仮想 IP アドレスをサブネットに含める必要があります。

18.2. インフラストラクチャー CR への障害ドメインの追加

既存の Nutanix クラスターに障害ドメインを追加するには、そのインフラストラクチャーカスタムリソース (CR) (**infrastructures.config.openshift.io**) を変更します。

ヒント

高可用性を確保するには、3 つの障害ドメインを設定することを推奨します。

手順

1. 次のコマンドを実行して、インフラストラクチャー CR を編集します。

```
$ oc edit infrastructures.config.openshift.io cluster
```

2. 障害ドメインを設定します。

Nutanix 障害ドメインを使用したインフラストラクチャー CR の例

```
spec:
  cloudConfig:
    key: config
    name: cloud-provider-config
#...
```

```
  platformSpec:
    nutanix:
      failureDomains:
      - cluster:
          type: UUID
          uuid: <uuid>
        name: <failure_domain_name>
        subnets:
        - type: UUID
          uuid: <network_uuid>
      - cluster:
          type: UUID
          uuid: <uuid>
        name: <failure_domain_name>
        subnets:
        - type: UUID
          uuid: <network_uuid>
      - cluster:
          type: UUID
          uuid: <uuid>
        name: <failure_domain_name>
        subnets:
        - type: UUID
          uuid: <network_uuid>
# ...
```

ここでは、以下のようになります。

<uuid>

Prism Element の汎用一意識別子 (UUID) を指定します。

<failure_domain_name>

障害ドメインの一意の名前を指定します。名前は 64 文字以下に制限されており、小文字、数字、ダッシュ (-) を含めることができます。ダッシュを名前の先頭または末尾に含めることはできません。

<network_uuid>

Prism Element サブネットオブジェクトの UUID を指定します。サブネットの IP アドレス接頭辞 (CIDR) には、OpenShift Container Platform クラスターが使用する仮想 IP アドレスを含める必要があります。OpenShift Container Platform クラスター内の障害ドメイン (Prism Element) ごとに 1 つのサブネットのみがサポートされます。

3. CR を保存して変更を適用します。

18.3. 障害ドメイン全体へのコントロールプレーンの分散

コントロールプレーンマシンセットのカスタムリソース (CR) を変更することで、Nutanix 障害ドメイン全体にコントロールプレーンを分散します。

前提条件

- クラスターのインフラストラクチャーカスタムリソース (CR) で障害ドメインを設定している。
- コントロールプレーンマシンセットのカスタムリソース (CR) がアクティブな状態である。

コントロールプレーンマシンセットのカスタムリソースの状態を確認する方法の詳細は、「関連情報」を参照してください。

手順

1. 次のコマンドを実行して、コントロールプレーンマシンセット CR を編集します。

```
$ oc edit controlplanemachineset.machine.openshift.io cluster -n openshift-machine-api
```

2. **spec.template.machines_v1beta1_machine_openshift_io.failureDomains** スタンザを追加して、障害ドメインを使用するようにコントロールプレーンマシンセットを設定します。

Nutanix 障害ドメインが設定されたコントロールプレーンマシンの例

```
apiVersion: machine.openshift.io/v1
kind: ControlPlaneMachineSet
  metadata:
    creationTimestamp: null
    labels:
      machine.openshift.io/cluster-api-cluster: <cluster_name>
    name: cluster
    namespace: openshift-machine-api
spec:
# ...
  template:
    machineType: machines_v1beta1_machine_openshift_io
    machines_v1beta1_machine_openshift_io:
      failureDomains:
        platform: Nutanix
        nutanix:
        - name: <failure_domain_name_1>
        - name: <failure_domain_name_2>
        - name: <failure_domain_name_3>
# ...
```

3. 変更を保存します。

デフォルトでは、コントロールプレーンマシンセットは、変更をコントロールプレーン設定に自動的に反映します。クラスターが **OnDelete** 更新ストラテジーを使用するように設定されている場合は、コントロールプレーンを手動で置き換える必要があります。詳細は、「関連情報」を参照してください。

関連情報

- コントロールプレーンマシンセットのカスタムリソースの状態を確認する
- コントロールプレーンマシンの置き換え

18.4. 障害ドメイン全体へのコンピュートマシンの分散

次のいずれかの方法で、Nutanix 障害ドメイン全体にコンピュートマシンを分散できます。

- 既存のコンピュートマシンセットを編集 すると、設定更新を最小限に抑えながら、Nutanix 障害ドメイン全体にコンピュートマシンを分散できます。

- 既存のコンピュートマシンセットを置き換える と、仕様をイミュータブルにして、すべてのマシンを確実に同じにすることができます。

18.4.1. コンピュートマシンセットの編集による障害ドメインの実装

既存のコンピュートマシンセットを使用して Nutanix 障害ドメイン全体にコンピュートマシンを分散するには、設定でコンピュートマシンセットを更新し、スケーリングを使用して既存のコンピュートマシンを置き換えます。

前提条件

- クラスターのインフラストラクチャーカスタムリソース (CR) で障害ドメインを設定している。

手順

1. 次のコマンドを実行して、クラスターのインフラストラクチャー CR を表示します。

```
$ oc describe infrastructures.config.openshift.io cluster
```

2. 各障害ドメイン (**platformSpec.nutanix.failureDomains**) について、クラスターの UUID、名前、サブネットオブジェクト UUID をメモします。これらの値は、障害ドメインをコンピュートマシンセットに追加するために必要です。

3. 以下のコマンドを実行して、クラスター内のコンピュートマシンセットを一覧表示します。

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

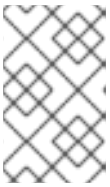
出力例

```
NAME                DESIRED   CURRENT   READY   AVAILABLE   AGE
<machine_set_name_1>  1       1         1       1           55m
<machine_set_name_2>  1       1         1       1           55m
```

4. 次のコマンドを実行して、最初のコンピュートマシンセットを編集します。

```
$ oc edit machineset <machine_set_name_1> -n openshift-machine-api
```

5. **spec.template.spec.providerSpec.value** スタンザを次のように更新して、最初の障害ドメインを使用するようにコンピュートマシンセットを設定します。



注記

cluster および **subnets** フィールドに指定する値が、クラスターのインフラストラクチャー CR の **failureDomains** スタンザに設定されている値と一致していることを確認してください。

Nutanix 障害ドメインを使用したコンピュートマシンセットの例

```
apiVersion: machine.openshift.io/v1
kind: MachineSet
metadata:
  creationTimestamp: null
  labels:
```

```
    machine.openshift.io/cluster-api-cluster: <cluster_name>
  name: <machine_set_name_1>
  namespace: openshift-machine-api
spec:
  replicas: 2
# ...
  template:
    spec:
# ...
      providerSpec:
        value:
          apiVersion: machine.openshift.io/v1
          failureDomain:
            name: <failure_domain_name_1>
          cluster:
            type: uuid
            uuid: <prism_element_uuid_1>
          subnets:
          - type: uuid
            uuid: <prism_element_network_uuid_1>
# ...
```

6. **spec.replicas** の値をメモします。この値は、変更を適用するためにコンピュートマシンセットをスケーリングする際に必要になるためです。

7. 変更を保存します。

8. 次のコマンドを実行して、更新されたコンピュートマシンセットによって管理されているマシンをリスト表示します。

```
$ oc get -n openshift-machine-api machines \
  -l machine.openshift.io/cluster-api-machineset=<machine_set_name_1>
```

出力例

```
NAME                        PHASE     TYPE   REGION    ZONE            AGE
<machine_name_original_1>   Running   AHV    Unnamed   Development-STS   4h
<machine_name_original_2>   Running   AHV    Unnamed   Development-STS   4h
```

9. 次のコマンドを実行して、更新されたコンピュートマシンセットで管理されるマシンごとに **delete** アノテーションを設定します。

```
$ oc annotate machine/<machine_name_original_1> \
  -n openshift-machine-api \
  machine.openshift.io/delete-machine="true"
```

10. 代わりとなるマシンを新しい設定で作成するために、次のコマンドを実行して、コンピュートマシンセットをレプリカ数の 2 倍にスケーリングします。

```
$ oc scale --replicas=<twice_the_number_of_replicas> \ (1)
  machineset <machine_set_name_1> \
  -n openshift-machine-api
```

1 たとえば、コンピュートマシンセット内のレプリカの元の数が **2** の場合、レプリカを **4** にスケーリングします。

11. 次のコマンドを実行して、更新されたコンピュートマシンセットによって管理されているマシンをリスト表示します。

```
$ oc get -n openshift-machine-api machines -l machine.openshift.io/cluster-api-machineset=
<machine_set_name_1>
```

新しいマシンが **Running** フェーズにある場合、コンピュートマシンセットを元のレプリカ数にスケーリングできます。

12. 古い設定で作成されたマシンを削除するために、次のコマンドを実行して、コンピュートマシンセットを元のレプリカ数にスケーリングします。

```
$ oc scale --replicas=<original_number_of_replicas> \ 1
  machineset <machine_set_name_1> \
  -n openshift-machine-api
```

1 たとえば、コンピュートマシンセット内のレプリカの元の数が **2** であった場合、レプリカを **2** にスケーリングします。

13. 必要に応じて、デプロイメントで使用可能な追加の障害ドメインを参照するようにマシンセットの変更を続けます。

関連情報

- コンピュートマシンセットの変更

18.4.2. コンピュートマシンセットの置き換えによる障害ドメインの実装

コンピュートマシンセットを置き換えることによって、Nutanix 障害ドメイン全体にコンピュートマシンを分散するには、独自の設定で新しいコンピュートマシンセットを作成し、作成されたマシンが起動するのを待ってから、古いコンピュートマシンセットを削除します。

前提条件

- クラスターのインフラストラクチャーカスタムリソース (CR) で障害ドメインを設定している。

手順

1. 次のコマンドを実行して、クラスターのインフラストラクチャー CR を表示します。

```
$ oc describe infrastructures.config.openshift.io cluster
```

2. 各障害ドメイン (**platformSpec.nutanix.failureDomains**) について、クラスターの UUID、名前、サブネットオブジェクト UUID をメモします。これらの値は、障害ドメインをコンピュートマシンセットに追加するために必要です。

3. 以下のコマンドを実行して、クラスター内のコンピュートマシンセットを一覧表示します。

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

出力例

```
NAME                        DESIRED   CURRENT   READY   AVAILABLE   AGE
<original_machine_set_name_1>  1         1         1       1          55m
<original_machine_set_name_2>  1         1         1       1          55m
```

4. 既存のコンピュートマシンセットの名前に注意してください。

5. 次のいずれかの方法を使用して、新しいコンピュートマシンセットのカスタムリソース (CR) の値を含む YAML ファイルを作成します。

   - 次のコマンドを実行して、既存のコンピュートマシンセット設定を新しいファイルにコピーします。

     ```
     $ oc get machineset <original_machine_set_name_1> \
       -n openshift-machine-api -o yaml > <new_machine_set_name_1>.yaml
     ```

     この YAML ファイルは、任意のテキストエディターで編集できます。

   - 任意のテキストエディターを使用して **<new_machine_set_name_1>.yaml** という名前の空の YAML ファイルを作成し、新しいコンピュートマシンセットに必要な値を含めます。特定のフィールドに設定する値がわからない場合は、次のコマンドを実行して、既存のコンピュートマシンセット CR の値を確認できます。

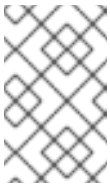
     ```
     $ oc get machineset <original_machine_set_name_1> \
       -n openshift-machine-api -o yaml
     ```

     出力例

     ```
     apiVersion: machine.openshift.io/v1beta1
     kind: MachineSet
     metadata:
       labels:
         machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
       name: <infrastructure_id>-<role> 2
       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: 3
             ...
     ```

     1 クラスターインフラストラクチャー ID。

2 デフォルトのノードラベル。



注記

user-provisioned infrastructure を持つクラスターの場合、コンピュートマシンセットが作成できるのは、**worker** または **infra** ロールを持つマシンのみです。

3 コンピュートマシンセット 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トマシンセット CR 設定を参照してください。

6. **<new_machine_set_name_1>.yaml** ファイルの **spec.template.spec.providerSpec.value** スタンザを更新または追加して、最初の障害ドメインを使用するように新しいコンピュートマシンセットを設定します。

注記

cluster および **subnets** フィールドに指定する値が、クラスターのインフラストラクチャー CR の **failureDomains** スタンザに設定されている値と一致していることを確認してください。

Nutanix 障害ドメインを使用したコンピュートマシンセットの例

```
apiVersion: machine.openshift.io/v1
kind: MachineSet
metadata:
  creationTimestamp: null
  labels:
    machine.openshift.io/cluster-api-cluster: <cluster_name>
  name: <new_machine_set_name_1>
  namespace: openshift-machine-api
spec:
  replicas: 2
# ...
  template:
    spec:
# ...
      providerSpec:
        value:
          apiVersion: machine.openshift.io/v1
          failureDomain:
            name: <failure_domain_name_1>
          cluster:
            type: uuid
            uuid: <prism_element_uuid_1>
          subnets:
          - type: uuid
            uuid: <prism_element_network_uuid_1>
# ...
```

7. 変更を保存します。

8. 次のコマンドを実行して、コンピュートマシンセット CR を作成します。

```
$ oc create -f <new_machine_set_name_1>.yaml
```

9. 必要に応じて、デプロイメントで使用可能な追加の障害ドメインを参照するコンピュートマシンセットの作成に進みます。

10. 新しいコンピュートマシンセットごとに次のコマンドを実行して、新しいコンピュートマシンセットによって管理されているマシンをリスト表示します。

```
$ oc get -n openshift-machine-api machines -l machine.openshift.io/cluster-api-machineset=
<new_machine_set_name_1>
```

出力例

```
NAME                        PHASE        TYPE   REGION    ZONE            AGE
<machine_from_new_1>        Provisioned   AHV    Unnamed   Development-STS   25s
<machine_from_new_2>        Provisioning  AHV    Unnamed   Development-STS   25s
```

新しいマシンが **Running** フェーズにある場合、障害ドメイン設定を含まない古いコンピュートマシンセットを削除できます。

11. 新しいマシンが **Running** フェーズにあることを確認したら、古いコンピュートマシンセットごとに次のコマンドを実行して削除します。

```
$ oc delete machineset <original_machine_set_name_1> -n openshift-machine-api
```

検証

- 設定を更新していないコンピューティングマシンセットが削除されたことを確認するには、次のコマンドを実行して、クラスター内のコンピューティングマシンセットをリスト表示します。

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

出力例

```
NAME                        DESIRED   CURRENT   READY   AVAILABLE   AGE
<new_machine_set_name_1>    1         1         1       1           4m12s
<new_machine_set_name_2>    1         1         1       1           4m12s
```

- 設定を更新していないコンピュートマシンが削除されたことを確認するには、次のコマンドを実行してクラスター内のマシンをリスト表示します。

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

削除中の出力例

```
NAME                        PHASE         TYPE     REGION     ZONE            AGE
<machine_from_new_1>        Running       AHV      Unnamed    Development-STS   5m41s
<machine_from_new_2>        Running       AHV      Unnamed    Development-STS   5m41s
```

```
<machine_from_original_1>  Deleting  AHV  Unnamed  Development-STS  4h
<machine_from_original_2>  Deleting  AHV  Unnamed  Development-STS  4h
```

削除完了時の出力例

```
NAME                    PHASE     TYPE  REGION    ZONE              AGE
<machine_from_new_1>    Running   AHV   Unnamed   Development-STS   6m30s
<machine_from_new_2>    Running   AHV   Unnamed   Development-STS   6m30s
```

- 新しいコンピューティングマシンセットによって作成されたマシンの設定が正しいことを確認するには、次のコマンドを実行して、いずれかの新しいマシンの CR に含まれる関連フィールドを調べます。

```
$ oc describe machine <machine_from_new_1> -n openshift-machine-api
```

関連情報

- Nutanix でコンピュートマシンセットを作成する

第19章 AWS LOCAL ZONE または WAVELENGTH ZONE 関連のタスク

OpenShift Container Platform を Amazon Web Services (AWS) にインストールした後、AWS Local Zones または Wavelength Zones とエッジコンピュートプールをさらに設定できます。

19.1. 既存のクラスターを拡張して AWS LOCAL ZONES または WAVELENGTH ZONES を使用する

インストール後のタスクとして、Amazon Web Services (AWS) 上の既存の OpenShift Container Platform クラスターを拡張して、AWS Local Zones または Wavelength Zones を使用できます。

ノードを Local Zones または Wavelength Zones の場所に拡張するには、次の手順を実行します。

- クラスターネットワークの最大伝送単位 (MTU) を調整します。
- Local Zones または Wavelength Zones グループを AWS Local Zones または Wavelength Zones にオプトインします。
- Local Zones または Wavelength Zones の場所の既存 VPC にサブネットを作成します。

重要

AWS 上の既存の OpenShift Container Platform クラスターを拡張して Local Zones または Wavelength Zones を使用する前に、既存の VPC に使用可能な Classless Inter-Domain Routing (CIDR) ブロックが含まれていることを確認してください。これらのブロックはサブネットの作成に必要です。

- マシンセットマニフェストを作成し、Local Zone または Wavelength Zone の各場所にノードを作成します。
- Local Zones のみ: **ec2:ModifyAvailabilityZoneGroup** 権限を Identity and Access Management (IAM) ユーザーまたはロールに追加して、必要なネットワークリソースを作成できるようにします。以下に例を示します。

  AWS Local Zones デプロイメントの追加 IAM ポリシーの例

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ec2:ModifyAvailabilityZoneGroup"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- Wavelength Zone のみ: **ec2:ModifyAvailabilityZoneGroup**、**ec2:CreateCarrierGateway**、および **ec2:DeleteCarrierGateway** 権限を Identity and Access Management (IAM) ユーザーまたはロールに追加して、必要なネットワークリソースを作成できるようにします。以下に例を示

します。

AWS Wavelength Zones デプロイメント用の追加 IAM ポリシーの例

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DeleteCarrierGateway",
        "ec2:CreateCarrierGateway"
      ],
      "Resource": "*"
    },
    {
      "Action": [
        "ec2:ModifyAvailabilityZoneGroup"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

関連情報

- AWS Local Zones、サポートされているインスタンスタイプ、およびサービスの詳細は、AWS ドキュメントの AWS Local Zones features を参照してください。
- AWS Local Zones、サポートされているインスタンスタイプ、およびサービスの詳細は、AWS ドキュメントの AWS Wavelength features を参照してください。

19.1.1. エッジコンピュートプールについて

エッジコンピュートノードは、AWS Local Zones または Wavelength Zone の場所で実行されるテイントされたコンピュートノードです。

Local Zones または Wavelength Zones を使用するクラスターをデプロイする場合は、次の点を考慮してください。

- Local Zones または Wavelength Zones 内の Amazon EC2 インスタンスは、アベイラビリティーゾーン内の Amazon EC2 インスタンスよりも高コストです。
- AWS Local Zones または Wavelength Zones で実行されているアプリケーションとエンドユーザーの間の遅延は低くなります。一部のワークロードでは、遅延の影響が発生します。たとえば、Local Zones または Wavelength Zones とアベイラビリティーゾーンの間で Ingress トラフィックが混在している場合などです。

重要

通常、Local Zones または Wavelength Zones 内の Amazon EC2 インスタンスとリージョン内の Amazon EC2 インスタンス間の最大伝送単位 (MTU) は 1300 です。クラスターネットワークの MTU は、オーバーヘッドを考慮して、常に EC2 の MTU よりも小さくする必要があります。具体的なオーバーヘッドは、ネットワークプラグインによって決まります。たとえば、OVN-Kubernetes のオーバーヘッドは **100 bytes** です。

ネットワークプラグインは、IPsec などの追加機能を提供できます。MTU のサイズには、このような追加機能も影響します。

それぞれのゾーンタイプの詳細については、次のリソースにアクセスしてください。

- AWS ドキュメントの How Local Zones work を参照してください。
- AWS ドキュメントの How AWS Wavelength work を参照してください。

OpenShift Container Platform 4.12 で、リモートゾーンで使用するために設計された新しいコンピュートプールの **エッジ** が導入されました。エッジコンピュートプールの設定は、AWS Local Zones または Wavelength Zones の場所間で共通です。Local Zones または Wavelength Zones リソース上の EC2 や EBS などのリソースのタイプとサイズの制限により、デフォルトのインスタンスタイプが従来のコンピュートプールと異なる場合があります。

Local Zones または Wavelength Zone の場所のデフォルト Elastic Block Store (EBS) は **gp2** であり、非エッジコンピュートプールとは異なります。各 Local Zones または Wavelength Zones に使用される、エッジコンピュートプールのインスタンスタイプも、ゾーンのインスタンスオファリングに応じて、他のコンピュートプールと異なる場合があります。

エッジコンピュートプールは、開発者が AWS Local Zones または Wavelength Zones ノードにアプリケーションをデプロイするために使用できる新しいラベルを作成します。新しいラベルは次のとおりです。

- **node-role.kubernetes.io/edge=''**
- Local Zones のみ: **machine.openshift.io/zone-type=local-zone**
- Wavelength Zones のみ: **machine.openshift.io/zone-type=wavelength-zone**
- **machine.openshift.io/zone-group=$ZONE_GROUP_NAME**

デフォルトでは、エッジコンピュートプールのマシンセットは **NoSchedule** テイントを定義して、Local Zones または Wavelength Zones のインスタンスに他のワークロードが拡散するのを防ぎます。ユーザーは、Pod 仕様で容認を定義している場合にのみユーザーワークロードを実行できます。

19.2. LOCAL ZONES または WAVELENGTH ZONES をサポートするためのクラスターネットワーク MTU の変更

場合によっては、クラスターインフラストラクチャーが Local Zones または Wavelength Zones のサブネットをサポートできるように、クラスターネットワークの最大伝送単位 (MTU) 値を変更する必要があります。

19.2.1. クラスター MTU について

インストール中に、クラスターネットワークの最大伝送ユニット (MTU) は、クラスター内のノードのプライマリーネットワークインターフェイスの MTU をもとに、自動的に検出されます。通常、検出された MTU をオーバーライドする必要はありません。

以下のような理由でクラスターネットワークの MTU を変更する場合があります。

- クラスターのインストール中に検出された MTU が使用中のインフラストラクチャーに適していない
- クラスターインフラストラクチャーに異なる MTU が必要となった (例: パフォーマンスの最適化にさまざまな MTU を必要とするノードが追加された)。

OVN-Kubernetes クラスターネットワークプラグインのみが MTU 値の変更をサポートしています。

19.2.1.1. サービス中断に関する考慮事項

クラスターで MTU の変更を開始すると、次の動作が原因でサービスの可用性に影響を与える可能性があります。

- 新しい MTU への移行を完了するには、少なくとも 2 回のローリングリブートが必要です。この間、一部のノードは再起動するため使用できません。
- 特定のアプリケーションに、絶対 TCP タイムアウト間隔よりもタイムアウトの間隔が短いクラスターにデプロイされた場合など、MTU の変更中に中断が発生する可能性があります。

19.2.1.2. MTU 値の選択

MTU の移行を計画するときは、関連しているが異なる MTU 値を 2 つ考慮する必要があります。

- **ハードウェア MTU**: この MTU 値は、ネットワークインフラストラクチャーの詳細に基づいて設定されます。
- **クラスターネットワーク MTU**: この MTU 値は、クラスターネットワークオーバーレイのオーバーヘッドを考慮して、常にハードウェア MTU よりも小さくなります。具体的なオーバーヘッドは、ネットワークプラグインによって決まります。OVN-Kubernetes の場合、オーバーヘッドは **100** バイトです。

クラスターがノードごとに異なる MTU 値を必要とする場合は、クラスター内の任意のノードで使用される最小の MTU 値から、ネットワークプラグインのオーバーヘッド値を差し引く必要があります。たとえば、クラスター内の一部のノードでは MTU が **9001** であり、MTU が **1500** のクラスターもある場合には、この値を **1400** に設定する必要があります。

重要

ノードが受け入れられない MTU 値の選択を回避するには、**ip -d link** コマンドを使用して、ネットワークインターフェイスが受け入れる最大 MTU 値 (**maxmtu**) を確認します。

19.2.1.3. 移行プロセスの仕組み

以下の表は、プロセスのユーザーが開始する手順と、移行が応答として実行するアクション間を区分して移行プロセスを要約しています。

表19.1 クラスター MTU のライブマイグレーション

ユーザーが開始する手順	OpenShift Container Platform アクティビティー
Cluster Network Operator 設定で次の値を指定します。 • **spec.migration.mtu.machine.to** • **spec.migration.mtu.network.from** • **spec.migration.mtu.network.to**	**Cluster Network Operator (CNO)**: 各フィールドが有効な値に設定されていることを確認します。 • **mtu.machine.to**は、新しいハードウェア MTU、またはハードウェアの MTU が変更されていない場合は、現在のハードウェア MTU のいずれかに設定する必要があります。この値は一時的なものであり、移行プロセスの一部として使用されます。これとは別に、既存のハードウェア MTU 値とは異なるハードウェア MTU を指定する場合は、マシン設定、DHCP 設定、Linux カーネルコマンドラインなどの他の方法で永続化するように MTU を手動で設定する必要があります。 • **mtu.network.from** フィールドは、クラスターネットワークの現在の MTU である **network.status.clusterNetworkMTU** フィールドと同じである必要があります。 • **mtu.network.to**フィールドは、ターゲットクラスターネットワーク MTU に設定する必要があり、ネットワークプラグインのオーバーレイオーバーヘッドを考慮して、ハードウェア MTU よりも低くする必要があります。OVN-Kubernetes の場合、オーバーヘッドは **100** バイトです。 指定の値が有効な場合に、CNO は、クラスターネットワークの MTU が**mtu.network.to**フィールドの値に設定された新しい一時設定を書き出します。 **Machine Config Operator (MCO)**: クラスター内の各ノードのローリングリブートを実行します。
クラスター上のノードのプライマリーネットワークインターフェイスの MTU を再設定します。これを実現するには、次のようなさまざまな方法を使用できます。 • MTU を変更した新しい NetworkManager 接続プロファイルのデプロイ • DHCP サーバー設定による MTU の変更 • ブートパラメーターによる MTU の変更	該当なし
ネットワークプラグインの CNO 設定で **mtu** 値を設定し、**spec.migration** を **null** に設定します。	**Machine Config Operator (MCO)**: 新しい MTU 設定を使用して、クラスター内の各ノードのローリングリブートを実行します。

19.2.1.4. クラスターネットワーク MTU の変更

クラスター管理者は、クラスターの最大伝送単位 (MTU) を増減できます。

重要

移行には中断が伴うため、MTU 更新が有効になると、クラスター内のノードが一時的に使用できなくなる可能性があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つアカウントを使用してクラスターにアクセスできる。
- クラスターのターゲット MTU を特定している。OVN-Kubernetes ネットワークプラグインの MTU は、クラスター内の最小のハードウェア MTU 値から **100** を引いた値に設定する必要があります。

手順

1. クラスターネットワークの現在の MTU を取得するには、次のコマンドを入力します。

```
$ oc describe network.config cluster
```

出力例

```
...
Status:
  Cluster Network:
    Cidr:               10.217.0.0/22
    Host Prefix:        23
  Cluster Network MTU:  1400
  Network Type:         OVNKubernetes
  Service Network:
    10.217.4.0/23
...
```

2. MTU 移行を開始するには、次のコマンドを入力して移行設定を指定します。Machine Config Operator は、MTU の変更に備えて、クラスター内のノードをローリングリブートします。

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": <overlay_from>, "to": <overlay_to> } ,
"machine": { "to" : <machine_to> } } } } }'
```

ここでは、以下のようになります。

<overlay_from>

現在のクラスターネットワークの MTU 値を指定します。

<overlay_to>

クラスターネットワークのターゲット MTU を指定します。この値は、**<machine_to>** の値を基準にして設定します。OVN-Kubernetes の場合、この値は **<machine_to>** の値から **100** を引いた値である必要があります。

<machine_to>

基盤となるホストネットワークのプライマリーネットワークインターフェイスの MTU を指定します。

クラスター MTU を増やす例

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": 1400, "to": 9000 } , "machine": { "to" :
9100} } } } }'
```

3. Machine Config Operator (MCO) は、各マシン設定プール内のマシンを更新するときに、各ノードを 1 つずつ再起動します。すべてのノードが更新されるまで待機する必要があります。以下のコマンドを実行してマシン設定プールのステータスを確認します。

```
$ oc get machineconfigpools
```

正常に更新されたノードには、**UPDATED=true**、**UPDATING=false**、**DEGRADED=false** のステータスがあります。

注記

Machine Config Operator は、デフォルトでプールごとに 1 つずつマシンを更新するため、クラスターのサイズに応じて移行にかかる合計時間が増加します。

4. ホスト上の新規マシン設定のステータスを確認します。

   a. マシン設定の状態と適用されたマシン設定の名前をリスト表示するには、以下のコマンドを入力します。

   ```
   $ oc describe node | egrep "hostname|machineconfig"
   ```

   出力例

   ```
   kubernetes.io/hostname=master-0
   machineconfiguration.openshift.io/currentConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/desiredConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/reason:
   machineconfiguration.openshift.io/state: Done
   ```

   b. 以下のステートメントが true であることを確認します。

   - **machineconfiguration.openshift.io/state** フィールドの値は **Done** です。
   - **machineconfiguration.openshift.io/currentConfig** フィールドの値は、**machineconfiguration.openshift.io/desiredConfig** フィールドの値と等しくなります。

   c. マシン設定が正しいことを確認するには、以下のコマンドを入力します。

   ```
   $ oc get machineconfig <config_name> -o yaml | grep ExecStart
   ```

   <config_name> は **machineconfiguration.openshift.io/currentConfig** フィールドのマシン設定の名前です。

   マシン設定には、systemd 設定に以下の更新を含める必要があります。

   -

```
ExecStart=/usr/local/bin/mtu-migration.sh
```

5. MTU の移行を完了するために、OVN-Kubernetes ネットワークプラグインに対して次のコマンドを入力します。

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": null, "defaultNetwork":{ "ovnKubernetesConfig": { "mtu": <mtu> }}}}'
```

ここでは、以下のようになります。

<mtu>

<overlay_to> で指定した新しいクラスターネットワーク MTU を指定します。

6. MTU の移行が完了すると、各マシン設定プールノードが 1 つずつ再起動します。すべてのノードが更新されるまで待機する必要があります。以下のコマンドを実行してマシン設定プールのステータスを確認します。

```
$ oc get machineconfigpools
```

正常に更新されたノードには、**UPDATED=true**、**UPDATING=false**、**DEGRADED=false** のステータスがあります。

検証

- 次のコマンドを入力して、クラスター内のノードが指定した MTU を使用していることを確認します。

```
$ oc describe network.config cluster
```

19.2.2. AWS Local Zones または Wavelength Zones へのオプトイン

AWS Local Zones または Wavelength Zones にサブネットを作成する予定がある場合は、各ゾーングループに個別にオプトインする必要があります。

前提条件

- AWS CLI をインストールしている。
- OpenShift Container Platform クラスターをデプロイする AWS リージョンを決定しました。
- ゾーングループにオプトインするユーザーまたはロールアカウントに、寛容な IAM ポリシーをアタッチしました。

手順

1. 次のコマンドを実行して、AWS リージョンで利用可能なゾーンをリスト表示します。

AWS リージョンで利用可能な AWS Local Zones をリスト表示するコマンドの例

```
$ aws --region "<value_of_AWS_Region>" ec2 describe-availability-zones \
    --query 'AvailabilityZones[].[{ZoneName: ZoneName, GroupName: GroupName, Status:
OptInStatus}]' \
```

```
    --filters Name=zone-type,Values=local-zone \
    --all-availability-zones
```

AWS リージョンで利用可能な AWS Wavelength Zones をリストするコマンドの例

```
$ aws --region "<value_of_AWS_Region>" ec2 describe-availability-zones \
    --query 'AvailabilityZones[].[{ZoneName: ZoneName, GroupName: GroupName, Status:
OptInStatus}]' \
    --filters Name=zone-type,Values=wavelength-zone \
    --all-availability-zones
```

AWS リージョンによっては、利用可能なゾーンのリストが長くなる場合があります。このコマンドは次のフィールドを返します。

ZoneName

Local Zones または Wavelength Zones の名前。

GroupName

ゾーンを設定するグループ。リージョンにオプトインするには、この名前を保存しておきます。

Status

Local Zones または Wavelength Zones グループのステータス。ステータスが **not-opted-in** の場合は、次の手順で説明するように **GroupName** をオプトインする必要があります。

2. 次のコマンドを実行して、AWS アカウントのゾーングループにオプトインします。

```
$ aws ec2 modify-availability-zone-group \
    --group-name "<value_of_GroupName>" \ 1
    --opt-in-status opted-in
```

1 **<value_of_GroupName>** は、サブネットを作成する Local Zones または Wavelength Zones のグループの名前に置き換えます。

19.2.3. AWS Local Zones または Wavelength Zones を使用する既存の VPC にネットワーク要件を作成する

Machine API でリモートゾーンの場所に Amazon EC2 インスタンスを作成する場合は、Local Zones または Wavelength Zones の場所にサブネットを作成する必要があります。Ansible や Terraform などのプロビジョニングツールを使用して、既存の Virtual Private Cloud (VPC) にサブネットを作成できます。

要件を満たすように CloudFormation テンプレートを設定できます。次のサブセクションでは、CloudFormation テンプレートを使用して、AWS Local Zones または Wavelength Zones を使用するように既存の VPC を拡張するネットワーク要件を作成する手順を説明します。

ノードを Local Zones に拡張するには、次のリソースを作成する必要があります。

- 2 つの VPC サブネット: パブリックとプライベート。パブリックサブネットは、リージョン内の通常のアベイラビリティーゾーンのパブリックルートテーブルに関連付けます。プライベートサブネットは、指定のルートテーブル ID に関連付けます。

ノードを Wavelength Zones に拡張するには、次のリソースを作成する必要があります。

- 指定の VPC ID に関連付けられた 1 つの VPC キャリアーゲートウェイ。
- VPC キャリアーゲートウェイへのデフォルトルートエントリーを含む、Wavelength Zones の 1 つの VPC ルートテーブル。
- 2 つの VPC サブネット: パブリックとプライベート。パブリックサブネットは、AWS Wavelength Zones のパブリックルートテーブルに関連付けます。プライベートサブネットは、指定のルートテーブル ID に関連付けます。

重要

このドキュメントの CloudFormation テンプレートは、Wavelength Zones の NAT ゲートウェイの制限を考慮して、プライベートサブネットと指定のルートテーブル ID の関連付けのみをサポートしています。ルートテーブル ID は、AWS リージョン内の有効な NAT ゲートウェイに割り当てられます。

19.2.4. Wavelength Zones のみ: VPC キャリアーゲートウェイの作成

Wavelength Zones で実行される OpenShift Container Platform クラスターでパブリックサブネットを使用するには、キャリアーゲートウェイを作成し、キャリアーゲートウェイを VPC に関連付ける必要があります。サブネットは、ロードバランサーまたはエッジコンピュートノードをデプロイするのに役立ちます。

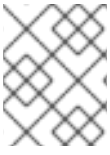
OpenShift Container Platform クラスター用の Wavelength Zones の場所に、エッジノードやインターネットに接続されたロードバランサーを作成するには、以下の必要なネットワークコンポーネントを作成する必要があります。

- 既存の VPC に関連付けるキャリアーゲートウェイ
- ルートエントリーをリストするキャリールートテーブル
- キャリアールートテーブルに関連付けるサブネット

キャリアーゲートウェイは、Wavelength Zone 内のサブネットのみを含む VPC に存在します。

以下では、AWS Wavelength Zones の場所に関連するキャリアーゲートウェイの機能を説明します。

- Wavelength Zone とキャリアーネットワーク (キャリアーネットワークから利用可能なデバイスを含む) の間の接続を提供します。
- ネットワークボーダーグループに格納されているパブリック IP アドレスである IP アドレスを Wavelength Zones からキャリアー IP アドレスに変換するなど、ネットワークアドレス変換 (NAT) 機能を実行します。このような変換機能は、受信トラフィックと送信トラフィックに適用されます。
- 特定の場所にあるキャリアーネットワークからの受信トラフィックを許可します。
- キャリアーネットワークとインターネットへの送信トラフィックを許可します。



注記

インターネットからキャリアーゲートウェイを経由した Wavelength Zone への受信接続設定は存在しません。

このドキュメントの CloudFormation テンプレートを使用して、次の AWS リソースのスタックを作成できます。

- テンプレート内の VPC ID に関連付ける 1 つのキャリアーゲートウェイ
- **<ClusterName>-public-carrier** という名前の Wavelength Zone の 1 つのパブリックルートテーブル
- キャリアーゲートウェイをターゲットとする新しいルートテーブルのデフォルトの IPv4 ルートエントリー
- AWS Simple Storage Service (S3) の VPC ゲートウェイエンドポイント

注記

このドキュメントの CloudFormation テンプレートを使用して AWS インフラストラクチャーを使用しない場合、記載されている情報を確認し、インフラストラクチャーを手動で作成する必要があります。クラスターが適切に初期化されない場合、インストールログを用意して Red Hat サポートに問い合わせする必要がある可能性があります。

前提条件

- AWS アカウントを設定している。
- **aws configure** を実行して、AWS キーおよびリージョンをローカルの AWS プロファイルに追加している。

手順

1. ドキュメントの次のセクション「VPC キャリアーゲートウェイ用の CloudFormation テンプレート」に移動し、**VPC キャリアーゲートウェイ用の CloudFormation テンプレート** から構文をコピーします。コピーしたテンプレートの構文を YAML ファイルとしてローカルシステムに保存します。このテンプレートは、クラスターに必要な VPC について記述しています。
2. 次のコマンドを実行して CloudFormation テンプレートをデプロイします。これにより、VPC を表す AWS リソースのスタックが作成されます。

```
$ aws cloudformation create-stack --stack-name <stack_name> \ 1
  --region ${CLUSTER_REGION} \
  --template-body file://<template>.yaml \ 2
  --parameters \//
    ParameterKey=VpcId,ParameterValue="${VpcId}" \ 3
    ParameterKey=ClusterName,ParameterValue="${ClusterName}" 4
```

1. **<stack_name>** は CloudFormation スタックの名前です (例: **clusterName-vpc-carrier-gw**)。クラスターを削除する場合に、このスタックの名前が必要になります。
2. **<template>** は、保存した CloudFormation テンプレート YAML ファイルの相対パスと名前です。
3. **<VpcId>** は、「AWS での VPC の作成」セクションで作成した CloudFormation スタックの出力から抽出した VPC ID です。
4. **<ClusterName>** は、CloudFormation スタックによって作成されるリソースに接頭辞として付加するカスタム値です。**install-config.yaml** 設定ファイルの **metadata.name** セクションで定義されているのと同じ名前を使用できます。

出力例

```
arn:aws:cloudformation:us-east-1:123456789012:stack/<stack_name>/dbedae40-2fd3-11eb-820e-12a48460849f
```

検証

- 次のコマンドを実行して、CloudFormation テンプレートコンポーネントが存在することを確認します。

```
$ aws cloudformation describe-stacks --stack-name <stack_name>
```

StackStatus に **CREATE_COMPLETE** が表示されると、出力に次のパラメーターの値が表示されます。このパラメーター値を、クラスターを作成するために実行する他の CloudFormation テンプレートに必ず指定してください。

PublicRouteTableId	キャリアーインフラストラクチャーのルートテーブルの ID。

19.2.5. Wavelength Zone のみ: VPC キャリアーゲートウェイ用の CloudFormation テンプレート

次の CloudFormation テンプレートを使用して、AWS Wavelength インフラストラクチャーにキャリアーゲートウェイをデプロイできます。

例19.1 VPC キャリアーゲートウェイ用の CloudFormation テンプレート

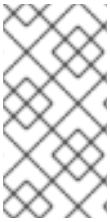
```
AWSTemplateFormatVersion: 2010-09-09
Description: Template for Creating Wavelength Zone Gateway (Carrier Gateway).

Parameters:
  VpcId:
    Description: VPC ID to associate the Carrier Gateway.
    Type: String
    AllowedPattern: ^(?:(?:vpc)(?:-[a-zA-Z0-9]+)?\b|(?:[0-9]{1,3}\.){3}[0-9]{1,3})$
    ConstraintDescription: VPC ID must be with valid name, starting with vpc-.*.
  ClusterName:
    Description: Cluster Name or Prefix name to prepend the tag Name for each subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: ClusterName parameter must be specified.

Resources:
  CarrierGateway:
    Type: "AWS::EC2::CarrierGateway"
    Properties:
      VpcId: !Ref VpcId
      Tags:
      - Key: Name
        Value: !Join ['-', [!Ref ClusterName, "cagw"]]
```

```
  PublicRouteTable:
    Type: "AWS::EC2::RouteTable"
    Properties:
      VpcId: !Ref VpcId
      Tags:
      - Key: Name
        Value: !Join ['-', [!Ref ClusterName, "public-carrier"]]

  PublicRoute:
    Type: "AWS::EC2::Route"
    DependsOn: CarrierGateway
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      CarrierGatewayId: !Ref CarrierGateway

  S3Endpoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      PolicyDocument:
        Version: 2012-10-17
        Statement:
        - Effect: Allow
          Principal: '*'
          Action:
          - '*'
          Resource:
          - '*'
      RouteTableIds:
      - !Ref PublicRouteTable
      ServiceName: !Join
      - ''
      - - com.amazonaws.
        - !Ref 'AWS::Region'
        - .s3
      VpcId: !Ref VpcId

Outputs:
  PublicRouteTableId:
    Description: Public Route table ID
    Value: !Ref PublicRouteTable
```

19.2.6. AWS エッジコンピュートサービス用のサブネットの作成

OpenShift Container Platform クラスターのエッジコンピュートノードのマシンセットを設定する前に、Local Zones または Wavelength Zones にサブネットを作成する必要があります。コンピュートノードをデプロイする Wavelength Zone ごとに次の手順を実行してください。

このドキュメントの CloudFormation テンプレートを使用して、CloudFormation スタックを作成できます。その後、このスタックを使用してサブネットをカスタムプロビジョニングできます。



注記

このドキュメントの CloudFormation テンプレートを使用して AWS インフラストラクチャーを使用しない場合、記載されている情報を確認し、インフラストラクチャーを手動で作成する必要があります。クラスターが適切に初期化されない場合、インストールログを用意して Red Hat サポートに問い合わせする必要がある可能性があります。

前提条件

- AWS アカウントを設定している。
- **aws configure** を実行して、AWS キーおよびリージョンをローカルの AWS プロファイルに追加している。
- Local Zones または Wavelength Zones グループにオプトインしている。

手順

1. このドキュメントの「VPC サブネット用の CloudFormation テンプレート」セクションに移動し、テンプレートから構文をコピーします。コピーしたテンプレートの構文を YAML ファイルとしてローカルシステムに保存します。このテンプレートは、クラスターに必要な VPC について記述しています。
2. 次のコマンドを実行して CloudFormation テンプレートをデプロイします。これにより、VPC を表す AWS リソースのスタックが作成されます。

```
$ aws cloudformation create-stack --stack-name <stack_name> \ 1
  --region ${CLUSTER_REGION} \
  --template-body file://<template>.yaml \ 2
  --parameters \
    ParameterKey=VpcId,ParameterValue="${VPC_ID}" \ 3
    ParameterKey=ClusterName,ParameterValue="${CLUSTER_NAME}" \ 4
    ParameterKey=ZoneName,ParameterValue="${ZONE_NAME}" \ 5
    ParameterKey=PublicRouteTableId,ParameterValue="${ROUTE_TABLE_PUB}" \ 6
    ParameterKey=PublicSubnetCidr,ParameterValue="${SUBNET_CIDR_PUB}" \ 7
    ParameterKey=PrivateRouteTableId,ParameterValue="${ROUTE_TABLE_PVT}" \ 8
    ParameterKey=PrivateSubnetCidr,ParameterValue="${SUBNET_CIDR_PVT}" 9
```

1. **<stack_name>** は、CloudFormation スタックの名前です。たとえば、Local Zones の場合は **cluster-wl-<local_zone_shortname>**、Wavelength Zones の場合は **cluster-wl-<wavelength_zone_shortname>** です。クラスターを削除する場合に、このスタックの名前が必要になります。
2. **<template>** は、保存した CloudFormation テンプレート YAML ファイルの相対パスと名前です。
3. **${VPC_ID}** は VPC ID であり、VPC 用の CloudFormation テンプレートの出力に含まれる値 **VpcID** です。
4. **${CLUSTER_NAME}** は、新しい AWS リソース名の接頭辞として使用する **ClusterName** の値です。
5. **${ZONE_NAME}** は、サブネットを作成する Local Zones または Wavelength Zones 名の値です。

6 **${ROUTE_TABLE_PUB}** は、CloudFormation テンプレートから抽出したパブリックルートテーブル ID です。Local Zones の場合、パブリックルートテーブルは VPC の

7 **${SUBNET_CIDR_PUB}** は、パブリックサブネットの作成に使用する有効な CIDR ブロックです。このブロックは、VPC CIDR ブロック **VpcCidr** の一部である必要があります。

8 **${ROUTE_TABLE_PVT}** は、VPC の CloudFormation スタックの出力から抽出した **PrivateRouteTableId** です。

9 **${SUBNET_CIDR_PVT}** は、プライベートサブネットの作成に使用する有効な CIDR ブロックです。このブロックは、VPC CIDR ブロック **VpcCidr** の一部である必要があります。

出力例

```
arn:aws:cloudformation:us-east-1:123456789012:stack/<stack_name>/dbedae40-820e-11eb-2fd3-12a48460849f
```

検証

- 次のコマンドを実行して、テンプレートコンポーネントが存在することを確認します。

```
$ aws cloudformation describe-stacks --stack-name <stack_name>
```

StackStatus に **CREATE_COMPLETE** が表示されると、出力に次のパラメーターの値が表示されます。

PublicSubnetId	CloudFormation スタックによって作成されたパブリックサブネットの ID。
PrivateSubnetId	CloudFormation スタックによって作成されたプライベートサブネットの ID。

これらのパラメーター値を、クラスターを作成するために実行する他の CloudFormation テンプレートに必ず指定してください。

19.2.7. VPC サブネット用の CloudFormation テンプレート

次の CloudFormation テンプレートを使用して、Local Zones または Wavelength Zones インフラストラクチャー上のゾーンにプライベートサブネットとパブリックサブネットをデプロイできます。

例19.2 VPC サブネット用の CloudFormation テンプレート

```
AWSTemplateFormatVersion: 2010-09-09
Description: Template for Best Practice Subnets (Public and Private)

Parameters:
  VpcId:
    Description: VPC ID that comprises all the target subnets.
    Type: String
    AllowedPattern: ^(?:(?:vpc)(?:-[a-zA-Z0-9]+)?\b|(?:[0-9]{1,3}\.){3}[0-9]{1,3})$
    ConstraintDescription: VPC ID must be with valid name, starting with vpc-.*.
```

```
  ClusterName:
    Description: Cluster name or prefix name to prepend the Name tag for each subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: ClusterName parameter must be specified.
  ZoneName:
    Description: Zone Name to create the subnets, such as us-west-2-lax-1a.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: ZoneName parameter must be specified.
  PublicRouteTableId:
    Description: Public Route Table ID to associate the public subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: PublicRouteTableId parameter must be specified.
  PublicSubnetCidr:
    AllowedPattern: ^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-
4][0-9]|25[0-5])(\/(1[6-9]|2[0-4]))$
    ConstraintDescription: CIDR block parameter must be in the form x.x.x.x/16-24.
    Default: 10.0.128.0/20
    Description: CIDR block for public subnet.
    Type: String
  PrivateRouteTableId:
    Description: Private Route Table ID to associate the private subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: PrivateRouteTableId parameter must be specified.
  PrivateSubnetCidr:
    AllowedPattern: ^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-
4][0-9]|25[0-5])(\/(1[6-9]|2[0-4]))$
    ConstraintDescription: CIDR block parameter must be in the form x.x.x.x/16-24.
    Default: 10.0.128.0/20
    Description: CIDR block for private subnet.
    Type: String


Resources:
  PublicSubnet:
    Type: "AWS::EC2::Subnet"
    Properties:
      VpcId: !Ref VpcId
      CidrBlock: !Ref PublicSubnetCidr
      AvailabilityZone: !Ref ZoneName
      Tags:
      - Key: Name
        Value: !Join ['-', [!Ref ClusterName, "public", !Ref ZoneName]]

  PublicSubnetRouteTableAssociation:
    Type: "AWS::EC2::SubnetRouteTableAssociation"
    Properties:
      SubnetId: !Ref PublicSubnet
      RouteTableId: !Ref PublicRouteTableId

  PrivateSubnet:
    Type: "AWS::EC2::Subnet"
    Properties:
```

```
      VpcId: !Ref VpcId
      CidrBlock: !Ref PrivateSubnetCidr
      AvailabilityZone: !Ref ZoneName
      Tags:
      - Key: Name
        Value: !Join ['-', [!Ref ClusterName, "private", !Ref ZoneName]]

  PrivateSubnetRouteTableAssociation:
    Type: "AWS::EC2::SubnetRouteTableAssociation"
    Properties:
      SubnetId: !Ref PrivateSubnet
      RouteTableId: !Ref PrivateRouteTableId

Outputs:
  PublicSubnetId:
    Description: Subnet ID of the public subnets.
    Value:
      !Join ["", [!Ref PublicSubnet]]

  PrivateSubnetId:
    Description: Subnet ID of the private subnets.
    Value:
      !Join ["", [!Ref PrivateSubnet]]
```

19.2.8. AWS Local Zones または Wavelength Zones ノードのマシンセットマニフェストの作成

AWS Local Zones または Wavelength Zones にサブネットを作成した後、マシンセットマニフェストを作成できます。

インストールプログラムは、クラスターのインストール時に **edge** マシンプールに次のラベルを設定します。

- **machine.openshift.io/parent-zone-name: <value_of_ParentZoneName>**
- **machine.openshift.io/zone-group: <value_of_ZoneGroup>**
- **machine.openshift.io/zone-type: <value_of_ZoneType>**

次の手順では、**edge**コンピュートプール設定に一致するマシンセット設定を作成する方法を詳しく説明します。

前提条件

- AWS Local Zones または Wavelength Zones にサブネットを作成している。

手順

- AWS API を収集してマシンセットマニフェストを作成するときに、**edge** マシンプールのラベルを手動で保存します。このアクションを完了するには、コマンドラインインターフェイス (CLI) で次のコマンドを入力します。

```
$ aws ec2 describe-availability-zones --region <value_of_Region> \ 1
    --query 'AvailabilityZones[].{
```

```
ZoneName: ZoneName,
ParentZoneName: ParentZoneName,
GroupName: GroupName,
ZoneType: ZoneType}' \
  --filters Name=zone-name,Values=<value_of_ZoneName> \ 2
  --all-availability-zones
```

1 **<value_of_Region>** には、ゾーンのリージョンの名前を指定します。

2 **<value_of_ZoneName>** には、Local Zones または Wavelength Zones の名前を指定します。

Local Zone us-east-1-nyc-1a の出力例

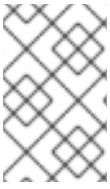
```
[
  {
    "ZoneName": "us-east-1-nyc-1a",
    "ParentZoneName": "us-east-1f",
    "GroupName": "us-east-1-nyc-1",
    "ZoneType": "local-zone"
  }
]
```

Wavelength Zone us-east-1-wl1 の出力例

```
[
  {
    "ZoneName": "us-east-1-wl1-bos-wlz-1",
    "ParentZoneName": "us-east-1a",
    "GroupName": "us-east-1-wl1",
    "ZoneType": "wavelength-zone"
  }
]
```

19.2.8.1. AWS 上のコンピュートマシンセットカスタムリソースのサンプル YAML

このサンプル YAML は **us-east-1-nyc-1a** Amazon Web Services (AWS) ゾーンで実行し、 **node-role.kubernetes.io/edge: ""** というラベルが付けられたノードを作成するコンピュートマシンセットを定義します。



注記

Wavelength Zone との関連でサンプル YAML ファイルを参照する場合は、必ず AWS リージョンとゾーンの情報をサポートされている Wavelength Zone の値に置き換えてください。

このサンプルでは、**<infrastructure_id>** はクラスターのプロビジョニング時に設定したクラスター ID に基づくインフラストラクチャー ID であり、**<edge>** は追加するノードラベルです。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
```

```
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-edge-<zone> 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id> 3
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-edge-<zone>
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id> 4
        machine.openshift.io/cluster-api-machine-role: edge 5
        machine.openshift.io/cluster-api-machine-type: edge 6
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-edge-<zone> 7
    spec:
      metadata:
        labels:
          machine.openshift.io/parent-zone-name: <value_of_ParentZoneName>
          machine.openshift.io/zone-group: <value_of_GroupName>
          machine.openshift.io/zone-type: <value_of_ZoneType>
          node-role.kubernetes.io/edge: "" 8
      providerSpec:
        value:
          ami:
            id: ami-046fe691f52a953f9 9
          apiVersion: machine.openshift.io/v1beta1
          blockDevices:
            - ebs:
                iops: 0
                volumeSize: 120
                volumeType: gp2
          credentialsSecret:
            name: aws-cloud-credentials
          deviceIndex: 0
          iamInstanceProfile:
            id: <infrastructure_id>-worker-profile 10
          instanceType: m6i.large
          kind: AWSMachineProviderConfig
          placement:
            availabilityZone: <zone> 11
            region: <region> 12
          securityGroups:
            - filters:
                - name: tag:Name
                  values:
                    - <infrastructure_id>-worker-sg 13
          subnet:
              id: <value_of_PublicSubnetIds> 14
          publicIp: true
          tags:
            - name: kubernetes.io/cluster/<infrastructure_id> 15
```

```
        value: owned
      - name: <custom_tag_name> 16
        value: <custom_tag_value> 17
    userDataSecret:
      name: worker-user-data
  taints: 18
    - key: node-role.kubernetes.io/edge
      effect: NoSchedule
```

1 3 4 10 13 15 クラスターのプロビジョニング時に設定したクラスター ID を基にするインフラストラクチャー ID を指定します。OpenShift CLI がインストールされている場合は、以下のコマンドを実行してインフラストラクチャー ID を取得できます。

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructure cluster
```

2 7 インフラストラクチャー ID、**edge** ロールノードラベル、およびゾーン名を指定します。

5 6 8 **edge** ロールノードラベルを指定します。

9 OpenShift Container Platform ノードの AWS ゾーンに有効な Red Hat Enterprise Linux CoreOS (RHCOS) Amazon Machine Image (AMI) を指定します。AWS Marketplace イメージを使用する場合は、AWS Marketplace から OpenShift Container Platform サブスクリプションを完了して、リージョンの AMI ID を取得する必要があります。

```
$ oc -n openshift-machine-api \
    -o jsonpath='{.spec.template.spec.providerSpec.value.ami.id}{"\n"}' \
    get machineset/<infrastructure_id>-<role>-<zone>
```

16 17 オプション: クラスターのカスタムタグデータを指定します。たとえば、**name:value** のペアである **Email:admin-email@example.com** を指定して、管理者の連絡先電子メールアドレスを追加できます。

注記

カスタムタグは、インストール中に **install-config.yml** ファイルで指定することもできます。**install-config.yml** ファイルとマシンセットに同じ **名前** のデータを持つタグが含まれている場合、マシンセットのタグの値が **install-config.yml** ファイルのタグの値よりも優先されます。

11 ゾーン名を指定します (例: **us-east-1-nyc-1a**)。

12 リージョン (例: **us-east-1**) を指定します。

14 AWS Local Zones または Wavelength Zones に作成したパブリックサブネットの ID。このパブリックサブネット ID は、「AWS ゾーンでのサブネットの作成」手順を完了したときに作成したものです。

18 ユーザーのワークロードが **edge** ノードにスケジュールされないようにテイントを指定します。

注記

インフラストラクチャーノードに **NoSchedule** テイントを追加すると、そのノードで実行されている既存の DNS Pod は **misscheduled** としてマークされます。**misscheduled** DNS Pod に対する容認の追加 または削除を行う必要があります。

19.2.8.2. コンピュート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. コンピュートマシンセットのカスタムリソース (CR) サンプルを含む新しい YAML ファイルを作成し、**<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ト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. 特定のコンピュートマシンセットカスタムリソース (CR) 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> 1
  name: <infrastructure_id>-<role> 2
  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: 3
        ...
```

1. クラスターインフラストラクチャー ID。
2. デフォルトのノードラベル。

注記

user-provisioned infrastructure を持つクラスターの場合、コンピュートマシンセットは **worker** および **infra** タイプのマシンのみを作成できます。

3. コンピュートマシンセット 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トマシンセット CR 設定を参照してください。

3. 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、コンピュート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-edge-us-east-1-nyc-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
```

新しいコンピュートマシンセットが利用可能になると、**DESIRED** と **CURRENT** の値が一致します。コンピュートマシンセットが使用できない場合は、数分待ってからコマンドを再実行してください。

- オプション: エッジマシンによって作成されたノードを確認するには、次のコマンドを実行します。

```
$ oc get nodes -l node-role.kubernetes.io/edge
```

出力例

```
NAME                          STATUS   ROLES         AGE    VERSION
ip-10-0-207-188.ec2.internal   Ready    edge,worker   172m   v1.25.2+d2e245f
```

関連情報

- AWS Local Zones 上のコンピュートノードを使用して AWS にクラスターをインストールする
- AWS Wavelength Zones 上のコンピュートノードを使用して AWS にクラスターをインストールする

19.3. AWS LOCAL ZONES または WAVELENGTH ZONES でのユーザーワークロードの作成

Amazon Web Service (AWS) の Local Zones または Wavelength Zone インフラストラクチャーを作成し、クラスターをデプロイした後、エッジコンピュートノードを使用して Local Zones または Wavelength Zones のサブネットにユーザーワークロードを作成できます。

インストールプログラムを使用してクラスターを作成すると、インストールプログラムは各エッジコンピュートノードに **NoSchedule** テイント effect を自動的に指定します。これは、Pod がテイントに対して指定された許容範囲に一致しない場合、スケジューラーは新しい Pod またはデプロイメントをノードに追加しないことを意味します。テイントを変更することで、Local Zones または Wavelength Zones の各サブネットにノードがワークロードを作成する方法をより適切に制御できます。

インストールプログラムは、**node-role.kubernetes.io/edge** ラベルと **node-role.kubernetes.io/worker** ラベルを含むコンピュートマシンセットのマニフェストファイルを作成します。これらのラベルは、Local Zones または Wavelength Zones のサブネット内にある各エッジコンピュートノードに適用されます。

注記

この手順の例は、Local Zones インフラストラクチャーを対象としています。Wavelength Zone インフラストラクチャーを使用している場合は、Wavelength Zone インフラストラクチャーでサポートされている機能に合わせて例を変更してください。

前提条件

- OpenShift CLI (**oc**) にアクセスできる。
- Local Zones または Wavelength Zones のサブネットが定義された Virtual Private Cloud (VPC) にクラスターをデプロイしている。
- Local Zones または Wavelength Zones のサブネット上のエッジコンピュートノードのコンピュートマシンセットが、**node-role.kubernetes.io/edge** のテイントを指定していることを確認している。

手順

1. Local Zones のサブネットで動作するエッジコンピュートノードにデプロイするサンプルアプリケーションの **deployment** リソース YAML ファイルを作成します。エッジコンピュートノードのテイントに合った正しい容認を指定していることを確認してください。

Local Zone のサブネットで動作するエッジコンピュートノード用に設定された deployment リソースの例

```
kind: Namespace
apiVersion: v1
metadata:
  name: <local_zone_application_namespace>
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc_name>
  namespace: <local_zone_application_namespace>
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: gp2-csi (1)
  volumeMode: Filesystem
---
apiVersion: apps/v1
kind: Deployment (2)
metadata:
  name: <local_zone_application> (3)
  namespace: <local_zone_application_namespace> (4)
spec:
  selector:
    matchLabels:
      app: <local_zone_application>
  replicas: 1
  template:
    metadata:
      labels:
        app: <local_zone_application>
        zone-group: ${ZONE_GROUP_NAME} (5)
    spec:
      securityContext:
        seccompProfile:
```

```
      type: RuntimeDefault
    nodeSelector: 6
      machine.openshift.io/zone-group: ${ZONE_GROUP_NAME}
    tolerations: 7
    - key: "node-role.kubernetes.io/edge"
      operator: "Equal"
      value: ""
      effect: "NoSchedule"
    containers:
      - image: openshift/origin-node
        command:
         - "/bin/socat"
        args:
          - TCP4-LISTEN:8080,reuseaddr,fork
          - EXEC:'/bin/bash -c \"printf \\\"HTTP/1.0 200 OK\r\n\r\n\\\"; sed -e \\\"/^\r/q\\\"\"'
        imagePullPolicy: Always
        name: echoserver
        ports:
          - containerPort: 8080
        volumeMounts:
          - mountPath: "/mnt/storage"
            name: data
    volumes:
    - name: data
      persistentVolumeClaim:
        claimName: <pvc_name>
```

1. **storageClassName**: Local Zone 設定の場合、 **gp2-csi** を指定する必要があります。
2. **kind**: **deployment** リソースを定義します。
3. **name**: Local Zone アプリケーションの名前を指定します。たとえば、 **local-zone-demo-app-nyc-1** です。
4. **namespace:** ユーザーワークロードを実行する AWS Local Zone 用の namespace を定義します。例: **local-zone-app-nyc-1a**
5. **zone-group**: ゾーンが属するグループを定義します。たとえば、**us-east-1-iah-1**
6. **nodeSelector**: 指定のラベルに一致するエッジコンピュートノードをターゲットとします。
7. **tolerations**: Local Zone ノードの **MachineSet** マニフェストで定義された **taints** と一致する値を設定します。

2. ノードの **service** リソース YAML ファイルを作成します。このリソースは、ターゲットのエッジコンピュートノードから Local Zone ネットワーク内で実行されるサービスに Pod を公開します。

Local Zone サブネットで動作するエッジコンピュートノード用に設定された service リソースの例

```
apiVersion: v1
kind: Service 1
metadata:
```

```
  name:  <local_zone_application>
  namespace: <local_zone_application_namespace>
spec:
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: NodePort
  selector: (2)
    app: <local_zone_application>
```

(1) **kind**: **service** リソースを定義します。

(2) **selector:** マネージド Pod に適用されるラベルタイプを指定します。

関連情報

- AWS Local Zones 上のコンピュートノードを使用して AWS にクラスターをインストールする
- AWS Wavelength Zones 上のコンピュートノードを使用して AWS にクラスターをインストールする
- テイントおよび容認 (Toleration) について

19.4. 次のステップ

- オプション: AWS Load Balancer (ALB) Operator を使用して、ターゲットのエッジコンピュートノードからパブリックネットワークの Local Zones または Wavelength Zones のサブネット内で実行されるサービスに Pod を公開します。AWS Load Balancer Operator のインストールを 参照してください。

第20章 AWS VPC クラスターの AWS OUTPOST への拡張

Amazon Web Services (AWS)のクラスターを既存の Amazon Virtual Private Cloud (VPC)にインストール した後に、AWS Outposts にコンピュートマシンをデプロイするコンピュートマシンセットを作成できます。AWS Outposts は、低遅延のオンプレミス環境とともに、クラウドベースの AWS デプロイメントの多くの機能を使用できるようにする AWS エッジコンピュートサービスです。詳細は、AWS Outposts のドキュメント を参照してください。

20.1. OPENSHIFT CONTAINER PLATFORM 上の AWS OUTPOSTS の要件と制限

以下の要件と制限に対応するように OpenShift Container Platform クラスターを設定すると、クラウドベースの AWS クラスター上のリソースと同様に AWS Outpost 上のリソースを管理できます。

- AWS 上の OpenShift Container Platform クラスターを Outpost に拡張するには、クラスターを既存の Amazon Virtual Private Cloud (VPC) にインストールしておく必要があります。
- Outpost のインフラストラクチャーは、AWS リージョンのアベイラビリティーゾーンに関連付けられており、専用のサブネットを使用します。Outpost にデプロイされた Edge コンピュートマシンは、Outpost のサブネットと、Outpost が関連付けられているアベイラビリティーゾーンを使用する必要があります。
- AWS Kubernetes クラウドコントローラーマネージャーは、Outpost サブネットを検出すると、Outpost サブネット内にサービスロードバランサーを作成しようとします。AWS Outposts は、サービスロードバランサーの実行をサポートしていません。クラウドコントローラーマネージャーが Outpost サブネットでサポート対象外のサービスを作成しないようにするには、Outpost サブネット設定に **kubernetes.io/cluster/unmanaged** タグを含める必要があります。この要件は、OpenShift Container Platform バージョン 4.16 における回避策です。詳細は、OCPBUGS-30041 を参照してください。
- AWS 上の OpenShift Container Platform クラスターには、**gp3-csi** および **gp2-csi** ストレージクラスがあります。これらのクラスは、Amazon Elastic Block Store (EBS) の gp3 および gp2 ボリュームに対応します。OpenShift Container Platform クラスターはデフォルトで **gp3-csi** ストレージクラスを使用しますが、AWS Outposts は EBS gp3 ボリュームをサポートしません。
- この実装では、**node-role.kubernetes.io/outposts** テイントを使用して、通常のクラスターのワークロードが Outpost ノードに分散するのを防ぎます。Outpost でユーザーのワークロードをスケジュールするには、アプリケーションの **Deployment** リソースで対応する容認を指定する必要があります。ユーザーのワークロード用に AWS Outpost インフラストラクチャーを予約すると、互換性を保つためにデフォルトの CSI を **gp2-csi** に更新するなど、追加の設定要件が回避されます。
- Outpost にボリュームを作成するには、CSI ドライバーに Outpost の Amazon Resource Name (ARN) が必要です。ドライバーは、**CSINode** オブジェクトに保存されているトポロジーキーを使用して、Outpost の ARN を特定します。ドライバーが正しいトポロジー値を使用するようにするには、ボリュームバインドモードを **WaitForConsumer** に設定した上で、作成する新しいストレージクラスに、許可されるトポロジーを設定しないようにする必要があります。
- AWS VPC クラスターを Outpost に拡張すると、2 種類のコンピューティングリソースが得られます。Outpost にはエッジコンピュートノードがあり、VPC にはクラウドベースのコンピュートノードがあります。クラウドベースの AWS Elastic Block ボリュームは、Outpost エッジコンピュートノードに接続できません。また、Outpost ボリュームはクラウドベースのコンピュートノードに接続できません。
  そのため、CSI スナップショットを使用して、永続ストレージを使用するアプリケーションを

クラウドベースのコンピュートノードからエッジコンピュートノードに移行したり、元の永続ボリュームを直接使用したりすることはできません。アプリケーションの永続ストレージデータを移行するには、手動でバックアップおよび復元操作を実行する必要があります。

- AWS Outposts は、AWS Network Load Balancer または AWS Classic Load Balancer をサポートしていません。AWS Outposts 環境でエッジコンピュートリソースの負荷分散を有効にするには、AWS Application Load Balancer を使用する必要があります。
  Application Load Balancer をプロビジョニングするには、Ingress リソースを使用し、AWS Load Balancer Operator をインストールする必要があります。クラスターに、ワークロードを共有するエッジベースとクラウドベースの両方のコンピュートインスタンスが含まれている場合は、追加の設定が必要です。

  詳細は、「Outpost に拡張された AWS VPC クラスターでの AWS Load Balancer Operator の使用」を参照してください。

関連情報

- Outpost に拡張された AWS VPC クラスターでの AWS Load Balancer Operator の使用

20.2. 環境に関する情報の取得

AWS VPC クラスターを Outpost に拡張するには、OpenShift Container Platform クラスターと Outpost 環境に関する情報を提供する必要があります。この情報を使用して、ネットワーク設定タスクを完了し、Outpost 内にコンピュートマシンを作成するコンピュートマシンセットを設定します。必要な詳細情報は、コマンドラインツールを使用して収集できます。

20.2.1. OpenShift Container Platform クラスターからの情報の取得

OpenShift CLI (**oc**) を使用して、OpenShift Container Platform クラスターから情報を取得できます。

ヒント

これらの値の一部またはすべてを、**export** コマンドを使用して環境変数として保存すると便利な場合があります。

前提条件

- OpenShift Container Platform クラスターを AWS のカスタム VPC にインストールしている。
- **cluster-admin** 権限を持つアカウントを使用してクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 次のコマンドを実行して、クラスターのインフラストラクチャー ID をリスト表示します。この値を保存しておきます。

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructures.config.openshift.io cluster
```

2. 次のコマンドを実行して、インストールプログラムが作成したコンピュートマシンセットに関する詳細を取得します。

a. クラスター上のコンピュートマシンセットをリスト表示します。

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

出力例

```
NAME                        DESIRED   CURRENT   READY   AVAILABLE   AGE
<compute_machine_set_name_1>  1       1         1       1           55m
<compute_machine_set_name_2>  1       1         1       1           55m
```

b. リストされたコンピュートマシンセットのいずれかの Amazon Machine Image (AMI) ID を表示します。この値を保存しておきます。

```
$ oc get machinesets.machine.openshift.io <compute_machine_set_name_1> \
  -n openshift-machine-api \
  -o jsonpath='{.spec.template.spec.providerSpec.value.ami.id}'
```

c. AWS VPC クラスターのサブネット ID を表示します。この値を保存しておきます。

```
$ oc get machinesets.machine.openshift.io <compute_machine_set_name_1> \
  -n openshift-machine-api \
  -o jsonpath='{.spec.template.spec.providerSpec.value.subnet.id}'
```

20.2.2. AWS アカウントからの情報の取得

AWS CLI (**aws**) を使用して、AWS アカウントから情報を取得できます。

ヒント

これらの値の一部またはすべてを、**export** コマンドを使用して環境変数として保存すると便利な場合があります。

前提条件

- 必要なハードウェアのセットアップが完了した AWS Outposts サイトがある。
- Outpost が AWS アカウントに接続されている。
- 必要なタスクを実行する権限を持つユーザーとして AWS CLI (**aws**) を使用して、AWS アカウントにアクセスできる。

手順

1. 次のコマンドを実行して、AWS アカウントに接続されている Outpost をリスト表示します。

```
$ aws outposts list-outposts
```

2. **aws outposts list-outposts** コマンドの出力の次の値を保存しておきます。
   - Outpost ID
   - Outpost の Amazon Resource Name (ARN)

- Outpost のアベイラビリティーゾーン

注記

aws outposts list-outposts コマンドの出力には、アベイラビリティーゾーンに関連する 2 つの値、**AvailabilityZone** と **AvailabilityZoneId** が含まれています。Outpost 内にコンピュートマシンを作成するコンピュートマシンセットを設定するには、**AvailablilityZone** 値を使用します。

3. Outpost ID の値を使用して次のコマンドを実行し、Outpost で利用可能なインスタンスタイプを表示します。利用可能なインスタンスタイプの値を保存しておきます。

```
$ aws outposts get-outpost-instance-types \
  --outpost-id <outpost_id_value>
```

4. Outpost ARN の値を使用して次のコマンドを実行し、Outpost のサブネット ID を表示します。この値を保存しておきます。

```
$ aws ec2 describe-subnets \
  --filters Name=outpost-arn,Values=<outpost_arn_value>
```

20.3. OUTPOST のネットワークの設定

VPC クラスターを Outpost に拡張するには、次のネットワーク設定タスクを完了する必要があります。

- クラスターネットワークの MTU を変更します。
- Outpost にサブネットを作成します。

20.3.1. AWS Outposts をサポートするためのクラスターネットワーク MTU の変更

クラスターネットワークの最大伝送単位 (MTU) は、インストール中に、クラスター内のノードのプライマリーネットワークインターフェイスの MTU に基づいて自動的に検出されます。場合によっては、AWS Outposts サブネットをサポートするために、クラスターネットワークの MTU 値を減らす必要があります。

重要

移行には中断が伴うため、MTU 更新が有効になると、クラスター内のノードが一時的に使用できなくなる可能性があります。

サービス中断に関する重要な考慮事項など、移行プロセスの詳細については、この手順の「関連情報」の「クラスターネットワークの MTU 変更」を参照してください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つアカウントを使用してクラスターにアクセスできる。

- クラスターのターゲット MTU を特定している。OVN-Kubernetes ネットワークプラグインの MTU は、クラスター内の最小のハードウェア MTU 値から **100** を引いた値に設定する必要があります。

手順

1. クラスターネットワークの現在の MTU を取得するには、次のコマンドを入力します。

```
$ oc describe network.config cluster
```

出力例

```
...
Status:
  Cluster Network:
    Cidr:               10.217.0.0/22
    Host Prefix:        23
  Cluster Network MTU:  1400
  Network Type:         OVNKubernetes
  Service Network:
    10.217.4.0/23
...
```

2. MTU 移行を開始するには、次のコマンドを入力して移行設定を指定します。Machine Config Operator は、MTU の変更に備えて、クラスター内のノードをローリングリブートします。

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": <overlay_from>, "to": <overlay_to> } , "machine": { "to" : <machine_to> } } } } }'
```

ここでは、以下のようになります。

<overlay_from>

現在のクラスターネットワークの MTU 値を指定します。

<overlay_to>

クラスターネットワークのターゲット MTU を指定します。この値は、**<machine_to>** の値を基準にして設定します。OVN-Kubernetes の場合、この値は **<machine_to>** の値から **100** を引いた値である必要があります。

<machine_to>

基盤となるホストネットワークのプライマリーネットワークインターフェイスの MTU を指定します。

クラスターの MTU を減らす例

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": 1400, "to": 1000 } , "machine": { "to" : 1100} } } } }'
```

3. Machine Config Operator (MCO) は、各マシン設定プール内のマシンを更新するときに、各ノードを 1 つずつ再起動します。すべてのノードが更新されるまで待機する必要があります。以下のコマンドを実行してマシン設定プールのステータスを確認します。

```
$ oc get machineconfigpools
```

正常に更新されたノードには、**UPDATED=true**、**UPDATING=false**、**DEGRADED=false** のステータスがあります。

注記

Machine Config Operator は、デフォルトでプールごとに 1 つずつマシンを更新するため、クラスターのサイズに応じて移行にかかる合計時間が増加します。

4. ホスト上の新規マシン設定のステータスを確認します。
   a. マシン設定の状態と適用されたマシン設定の名前をリスト表示するには、以下のコマンドを入力します。

```
$ oc describe node | egrep "hostname|machineconfig"
```

出力例

```
kubernetes.io/hostname=master-0
machineconfiguration.openshift.io/currentConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/desiredConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/reason:
machineconfiguration.openshift.io/state: Done
```

   b. 以下のステートメントが true であることを確認します。

- **machineconfiguration.openshift.io/state** フィールドの値は **Done** です。
- **machineconfiguration.openshift.io/currentConfig** フィールドの値は、**machineconfiguration.openshift.io/desiredConfig** フィールドの値と等しくなります。

   c. マシン設定が正しいことを確認するには、以下のコマンドを入力します。

```
$ oc get machineconfig <config_name> -o yaml | grep ExecStart
```

<config_name> は **machineconfiguration.openshift.io/currentConfig** フィールドのマシン設定の名前です。

マシン設定には、systemd 設定に以下の更新を含める必要があります。

```
ExecStart=/usr/local/bin/mtu-migration.sh
```

5. MTU の移行を完了するために、OVN-Kubernetes ネットワークプラグインに対して次のコマンドを入力します。

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": null, "defaultNetwork":{ "ovnKubernetesConfig": { "mtu": <mtu> }}}}'
```

ここでは、以下のようになります。

<mtu>

<overlay_to> で指定した新しいクラスターネットワーク MTU を指定します。

6. MTU の移行が完了すると、各マシン設定プールノードが 1 つずつ再起動します。すべてのノードが更新されるまで待機する必要があります。以下のコマンドを実行してマシン設定プールのステータスを確認します。

```
$ oc get machineconfigpools
```

正常に更新されたノードには、**UPDATED=true**、**UPDATING=false**、**DEGRADED=false** のステータスがあります。

検証

- 次のコマンドを入力して、クラスター内のノードが指定した MTU を使用していることを確認します。

```
$ oc describe network.config cluster
```

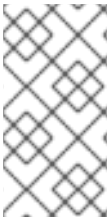
関連情報

- クラスターネットワークの MTU 変更

20.3.2. AWS エッジコンピュートサービス用のサブネットの作成

OpenShift Container Platform クラスターのエッジコンピュートノードのマシンセットを設定する前に、AWS Outposts にサブネットを作成する必要があります。

このドキュメントの CloudFormation テンプレートを使用して、CloudFormation スタックを作成できます。その後、このスタックを使用してサブネットをカスタムプロビジョニングできます。



注記

このドキュメントの CloudFormation テンプレートを使用して AWS インフラストラクチャーを使用しない場合、記載されている情報を確認し、インフラストラクチャーを手動で作成する必要があります。クラスターが適切に初期化されない場合、インストールログを用意して Red Hat サポートに問い合わせする必要がある可能性があります。

前提条件

- AWS アカウントを設定している。
- **aws configure** を実行して、AWS キーおよびリージョンをローカルの AWS プロファイルに追加している。
- OpenShift Container Platform クラスター、Outpost、および AWS アカウントから環境に関する必要な情報を取得している。

手順

1. このドキュメントの「VPC サブネット用の CloudFormation テンプレート」セクションに移動し、テンプレートから構文をコピーします。コピーしたテンプレートの構文を YAML ファイルとしてローカルシステムに保存します。このテンプレートは、クラスターに必要な VPC について記述しています。

2. 次のコマンドを実行して CloudFormation テンプレートをデプロイします。これにより、VPC を表す AWS リソースのスタックが作成されます。

```
$ aws cloudformation create-stack --stack-name <stack_name> \ 1
  --region ${CLUSTER_REGION} \
  --template-body file://<template>.yaml \ 2
  --parameters \
    ParameterKey=VpcId,ParameterValue="${VPC_ID}" \ 3
    ParameterKey=ClusterName,ParameterValue="${CLUSTER_NAME}" \ 4
    ParameterKey=ZoneName,ParameterValue="${ZONE_NAME}" \ 5
    ParameterKey=PublicRouteTableId,ParameterValue="${ROUTE_TABLE_PUB}" \ 6
    ParameterKey=PublicSubnetCidr,ParameterValue="${SUBNET_CIDR_PUB}" \ 7
    ParameterKey=PrivateRouteTableId,ParameterValue="${ROUTE_TABLE_PVT}" \ 8
    ParameterKey=PrivateSubnetCidr,ParameterValue="${SUBNET_CIDR_PVT}" \ 9
    ParameterKey=PrivateSubnetLabel,ParameterValue="private-outpost" \
    ParameterKey=PublicSubnetLabel,ParameterValue="public-outpost" \
    ParameterKey=OutpostArn,ParameterValue="${OUTPOST_ARN}" 10
```

1. **<stack_name>** は、CloudFormation スタックの名前です (**cluster-<outpost_name>** など)。
2. **<template>** は、保存した CloudFormation テンプレート YAML ファイルの相対パスと名前です。
3. **${VPC_ID}** は VPC ID であり、VPC 用の CloudFormation テンプレートの出力に含まれる値 **VpcID** です。
4. **${CLUSTER_NAME}** は、新しい AWS リソース名の接頭辞として使用する **ClusterName** の値です。
5. **${ZONE_NAME}** は、サブネットを作成する AWS Outposts 名の値です。
6. **${ROUTE_TABLE_PUB}** は、Outpost 上のパブリックサブネットを関連付けるために使用する、**${VPC_ID}** に作成されたパブリックルートテーブル ID です。このスタックによって作成された Outpost サブネットを関連付けるパブリックルートテーブルを指定します。
7. **${SUBNET_CIDR_PUB}** は、パブリックサブネットの作成に使用する有効な CIDR ブロックです。このブロックは、VPC CIDR ブロック **VpcCidr** の一部である必要があります。
8. **${ROUTE_TABLE_PVT}** は、Outpost 上のプライベートサブネットを関連付けるために使用する、**${VPC_ID}** に作成されたプライベートルートテーブル ID です。このスタックによって作成された Outpost サブネットを関連付けるプライベートルートテーブルを指定します。
9. **${SUBNET_CIDR_PVT}** は、プライベートサブネットの作成に使用する有効な CIDR ブロックです。このブロックは、VPC CIDR ブロック **VpcCidr** の一部である必要があります。
10. **${OUTPOST_ARN}** は、Outpost の Amazon Resource Name (ARN) です。

出力例

```
arn:aws:cloudformation:us-east-1:123456789012:stack/<stack_name>/dbedae40-820e-11eb-2fd3-12a48460849f
```

検証

- 次のコマンドを実行して、テンプレートコンポーネントが存在することを確認します。

```
$ aws cloudformation describe-stacks --stack-name <stack_name>
```

StackStatus に **CREATE_COMPLETE** が表示されると、出力に次のパラメーターの値が表示されます。

PublicSubnetId	CloudFormation スタックによって作成されたパブリックサブネットの ID。
PrivateSubnetId	CloudFormation スタックによって作成されたプライベートサブネットの ID。

これらのパラメーター値を、クラスターを作成するために実行する他の CloudFormation テンプレートに必ず指定してください。

20.3.3. VPC サブネット用の CloudFormation テンプレート

次の CloudFormation テンプレートを使用して、Outpost サブネットをデプロイできます。

例20.1 VPC サブネット用の CloudFormation テンプレート

```
AWSTemplateFormatVersion: 2010-09-09
Description: Template for Best Practice Subnets (Public and Private)

Parameters:
  VpcId:
    Description: VPC ID that comprises all the target subnets.
    Type: String
    AllowedPattern: ^(?:(?:vpc)(?:-[a-zA-Z0-9]+)?\b|(?:[0-9]{1,3}\.){3}[0-9]{1,3})$
    ConstraintDescription: VPC ID must be with valid name, starting with vpc-.*.
  ClusterName:
    Description: Cluster name or prefix name to prepend the Name tag for each subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: ClusterName parameter must be specified.
  ZoneName:
    Description: Zone Name to create the subnets, such as us-west-2-lax-1a.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: ZoneName parameter must be specified.
  PublicRouteTableId:
    Description: Public Route Table ID to associate the public subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: PublicRouteTableId parameter must be specified.
  PublicSubnetCidr:
```

```
    AllowedPattern: ^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-
4][0-9]|25[0-5])(\/(1[6-9]|2[0-4]))$
    ConstraintDescription: CIDR block parameter must be in the form x.x.x.x/16-24.
    Default: 10.0.128.0/20
    Description: CIDR block for public subnet.
    Type: String
  PrivateRouteTableId:
    Description: Private Route Table ID to associate the private subnet.
    Type: String
    AllowedPattern: ".+"
    ConstraintDescription: PrivateRouteTableId parameter must be specified.
  PrivateSubnetCidr:
    AllowedPattern: ^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-
4][0-9]|25[0-5])(\/(1[6-9]|2[0-4]))$
    ConstraintDescription: CIDR block parameter must be in the form x.x.x.x/16-24.
    Default: 10.0.128.0/20
    Description: CIDR block for private subnet.
    Type: String
  PrivateSubnetLabel:
    Default: "private"
    Description: Subnet label to be added when building the subnet name.
    Type: String
  PublicSubnetLabel:
    Default: "public"
    Description: Subnet label to be added when building the subnet name.
    Type: String
  OutpostArn:
    Default: ""
    Description: OutpostArn when creating subnets on AWS Outpost.
    Type: String

Conditions:
  OutpostEnabled: !Not [!Equals [!Ref "OutpostArn", ""]]

Resources:
  PublicSubnet:
    Type: "AWS::EC2::Subnet"
    Properties:
      VpcId: !Ref VpcId
      CidrBlock: !Ref PublicSubnetCidr
      AvailabilityZone: !Ref ZoneName
      OutpostArn: !If [ OutpostEnabled, !Ref OutpostArn, !Ref "AWS::NoValue"]
      Tags:
      - Key: Name
        Value: !Join ['-', [ !Ref ClusterName, !Ref PublicSubnetLabel, !Ref ZoneName]]
      - Key: kubernetes.io/cluster/unmanaged 1
        Value: true

  PublicSubnetRouteTableAssociation:
    Type: "AWS::EC2::SubnetRouteTableAssociation"
    Properties:
      SubnetId: !Ref PublicSubnet
      RouteTableId: !Ref PublicRouteTableId

  PrivateSubnet:
    Type: "AWS::EC2::Subnet"
```

```
      Properties:
        VpcId: !Ref VpcId
        CidrBlock: !Ref PrivateSubnetCidr
        AvailabilityZone: !Ref ZoneName
        OutpostArn: !If [ OutpostEnabled, !Ref OutpostArn, !Ref "AWS::NoValue"]
        Tags:
        - Key: Name
          Value: !Join ['-', [!Ref ClusterName, !Ref PrivateSubnetLabel, !Ref ZoneName]]
        - Key: kubernetes.io/cluster/unmanaged 2
          Value: true

    PrivateSubnetRouteTableAssociation:
      Type: "AWS::EC2::SubnetRouteTableAssociation"
      Properties:
        SubnetId: !Ref PrivateSubnet
        RouteTableId: !Ref PrivateRouteTableId

  Outputs:
    PublicSubnetId:
      Description: Subnet ID of the public subnets.
      Value:
        !Join ["", [!Ref PublicSubnet]]

    PrivateSubnetId:
      Description: Subnet ID of the private subnets.
      Value:
        !Join ["", [!Ref PrivateSubnet]]
```

1 AWS Outposts のパブリックサブネット設定に **kubernetes.io/cluster/unmanaged** タグを含める必要があります。

2 AWS Outposts のプライベートサブネット設定に **kubernetes.io/cluster/unmanaged** タグを含める必要があります。

20.4. OUTPOST にエッジコンピュートマシンをデプロイするコンピュートマシンセットの作成

AWS Outposts でエッジコンピュートマシンを作成するには、互換性のある設定を使用して新しいコンピュートマシンセットを作成する必要があります。

前提条件

- AWS Outposts サイトがある。
- OpenShift Container Platform クラスターを AWS のカスタム VPC にインストールしている。
- **cluster-admin** 権限を持つアカウントを使用してクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 以下のコマンドを実行して、クラスター内のコンピュートマシンセットを一覧表示します。

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

出力例

```
NAME                        DESIRED   CURRENT   READY   AVAILABLE   AGE
<original_machine_set_name_1>  1         1         1       1           55m
<original_machine_set_name_2>  1         1         1       1           55m
```

2. 既存のコンピュートマシンセットの名前を記録します。

3. 次のいずれかの方法を使用して、新しいコンピュートマシンセットのカスタムリソース (CR) の値を含む YAML ファイルを作成します。

   - 次のコマンドを実行して、既存のコンピュートマシンセット設定を新しいファイルにコピーします。

     ```
     $ oc get machinesets.machine.openshift.io <original_machine_set_name_1> \
       -n openshift-machine-api -o yaml > <new_machine_set_name_1>.yaml
     ```

     この YAML ファイルは、任意のテキストエディターで編集できます。

   - 任意のテキストエディターを使用して **<new_machine_set_name_1>.yaml** という名前の空の YAML ファイルを作成し、新しいコンピュートマシンセットに必要な値を含めます。特定のフィールドに設定する値がわからない場合は、次のコマンドを実行して、既存のコンピュートマシンセット CR の値を確認できます。

     ```
     $ oc get machinesets.machine.openshift.io <original_machine_set_name_1> \
       -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> 1
       name: <infrastructure_id>-<role>-<availability_zone> 2
       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>-
     <availability_zone>
       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>-
     <availability_zone>
     ```

```
    spec:
      providerSpec: 3
  # ...
```

1. クラスターインフラストラクチャー ID。
2. デフォルトのノードラベル。AWS Outposts の場合は、**outposts** ロールを使用します。
3. 省略されている **providerSpec** セクションには、Outpost 用に設定する必要がある値が含まれています。

4. **<new_machine_set_name_1>.yaml** ファイルを編集して、Outpost にエッジコンピュートマシンを作成するように新しいコンピュートマシンセットを設定します。

AWS Outposts 用のコンピュートマシンセットの例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-outposts-<availability_zone> 2
  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>-outposts-
<availability_zone>
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: outposts
        machine.openshift.io/cluster-api-machine-type: outposts
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-outposts-
<availability_zone>
    spec:
      metadata:
        labels:
          node-role.kubernetes.io/outposts: ""
          location: outposts
      providerSpec:
        value:
          ami:
            id: <ami_id> 3
          apiVersion: machine.openshift.io/v1beta1
          blockDevices:
            - ebs:
                volumeSize: 120
                volumeType: gp2 4
          credentialsSecret:
```

```
        name: aws-cloud-credentials
      deviceIndex: 0
      iamInstanceProfile:
        id: <infrastructure_id>-worker-profile
      instanceType: m5.xlarge 5
      kind: AWSMachineProviderConfig
      placement:
        availabilityZone: <availability_zone>
        region: <region> 6
      securityGroups:
        - filters:
            - name: tag:Name
              values:
                - <infrastructure_id>-worker-sg
      subnet:
        id: <subnet_id> 7
      tags:
        - name: kubernetes.io/cluster/<infrastructure_id>
          value: owned
      userDataSecret:
        name: worker-user-data
    taints: 8
      - key: node-role.kubernetes.io/outposts
        effect: NoSchedule
```

1. クラスターインフラストラクチャー ID を指定します。
2. コンピュートマシンセットの名前を指定します。この名前は、クラスターインフラストラクチャー ID、**outposts** ロール名、および Outpost のアベイラビリティーゾーンでされます。
3. Amazon Machine Image (AMI) ID を指定します。
4. EBS ボリュームのタイプを指定します。AWS Outposts には gp2 ボリュームが必要です。
5. AWS インスタンスのタイプを指定します。Outpost で設定されているインスタンスタイプを使用する必要があります。
6. Outpost のアベイラビリティーゾーンが存在する AWS リージョンを指定します。
7. Outpost の専用サブネットを指定します。
8. **node-role.kubernetes.io/outposts** ラベルを持つノードでワークロードがスケジュールされないようにテイントを指定します。Outpost でユーザーのワークロードをスケジュールするには、アプリケーションの **Deployment** リソースで対応する容認を指定する必要があります。

5. 変更を保存します。
6. 次のコマンドを実行して、コンピュートマシンセット CR を作成します。

```
$ oc create -f <new_machine_set_name_1>.yaml
```

検証

- コンピュートマシンセットが作成されたことを確認するには、次のコマンドを実行してクラスター内のコンピュートマシンセットをリスト表示します。

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

出力例

```
NAME                        DESIRED   CURRENT   READY   AVAILABLE   AGE
<new_machine_set_name_1>    1         1         1       1           4m12s
<original_machine_set_name_1>  1      1         1       1           55m
<original_machine_set_name_2>  1      1         1       1           55m
```

- 新しいコンピュートマシンセットによって管理されるマシンをリスト表示するには、次のコマンドを実行します。

```
$ oc get -n openshift-machine-api machines.machine.openshift.io \
  -l machine.openshift.io/cluster-api-machineset=<new_machine_set_name_1>
```

出力例

```
NAME                        PHASE         TYPE       REGION      ZONE         AGE
<machine_from_new_1>        Provisioned   m5.xlarge  us-east-1   us-east-1a   25s
<machine_from_new_2>        Provisioning  m5.xlarge  us-east-1   us-east-1a   25s
```

- 新しいコンピューティングマシンセットによって作成されたマシンの設定が正しいことを確認するには、次のコマンドを実行して、いずれかの新しいマシンの CR に含まれる関連フィールドを調べます。

```
$ oc describe machine <machine_from_new_1> -n openshift-machine-api
```

20.5. OUTPOST でのユーザーワークロードの作成

AWS VPC クラスター内の OpenShift Container Platform を Outpost に拡張した後、ラベル **node-role.kubernetes.io/outposts** を持つエッジコンピュートノードを使用して、Outpost にユーザーワークロードを作成できます。

前提条件

- AWS VPC クラスターを Outpost に拡張した。
- **cluster-admin** 権限を持つアカウントを使用してクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- Outpost 環境と互換性のあるエッジコンピュートマシンをデプロイするコンピュートマシンセットを作成している。

手順

1. エッジサブネットのエッジコンピュートノードにデプロイするアプリケーションの **Deployment** リソースファイルを設定します。

Deployment マニフェストの例

```
kind: Namespace
apiVersion: v1
metadata:
  name: <application_name> 1
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <application_name>
  namespace: <application_namespace> 2
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: gp2-csi 3
  volumeMode: Filesystem
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: <application_name>
  namespace: <application_namespace>
spec:
  selector:
    matchLabels:
      app: <application_name>
  replicas: 1
  template:
    metadata:
      labels:
        app: <application_name>
        location: outposts 4
    spec:
      securityContext:
        seccompProfile:
          type: RuntimeDefault
      nodeSelector: 5
        node-role.kubernetes.io/outpost: ''
      tolerations: 6
      - key: "node-role.kubernetes.io/outposts"
        operator: "Equal"
        value: ""
        effect: "NoSchedule"
      containers:
        - image: openshift/origin-node
          command:
           - "/bin/socat"
          args:
            - TCP4-LISTEN:8080,reuseaddr,fork
            - EXEC:'/bin/bash -c \"printf \\\"HTTP/1.0 200 OK\r\n\r\n\\\"; sed -e \\\"/^\r/q\\\"\"'
          imagePullPolicy: Always
          name: <application_name>
          ports:
```

```
        - containerPort: 8080
      volumeMounts:
        - mountPath: "/mnt/storage"
          name: data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: <application_name>
```

1. アプリケーションの名前を指定します。
2. アプリケーションの namespace を指定します。アプリケーションの namespace はアプリケーション名と同じにすることができます。
3. ストレージクラス名を指定します。エッジコンピュート設定の場合は、**gp2-csi** ストレージクラスを使用する必要があります。
4. Outpost にデプロイされるワークロードを識別するラベルを指定します。
5. エッジコンピュートノードをターゲットとするノードセレクターラベルを指定します。
6. エッジコンピュートマシンのコンピュートマシンセット内の **key** および **effects** テイントに一致する容認を指定します。この例のように **value** と **operator** の容認を設定します。

2. 次のコマンドを実行して、**Deployment** リソースを作成します。

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

3. ターゲットのエッジコンピュートノードからエッジネットワーク内で実行されるサービスに Pod を公開する **Service** オブジェクトを設定します。

Service マニフェストの例

```
apiVersion: v1
kind: Service 1
metadata:
  name:  <application_name>
  namespace: <application_namespace>
spec:
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: NodePort
  selector: 2
    app: <application_name>
```

1. **service** リソースを定義します。
2. 管理対象の Pod に適用するラベルのタイプを指定します。

4. 次のコマンドを実行して **Service** CR を作成します。

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

20.6. エッジおよびクラウドベースの AWS コンピューティングリソースでワークロードをスケジュールする

AWS VPC クラスターを Outpost に拡張すると、Outpost はエッジコンピュートノードを使用し、VPC はクラウドベースのコンピュートノードを使用します。Outpost に拡張された AWS VPC クラスターには、ロードバランサーに関する次の考慮事項が適用されます。

- Outpost は AWS Network Load Balancer または AWS Classic Load Balancer を実行できませんが、Outpost に拡張された VPC クラスターの Classic Load Balancer は Outpost エッジコンピュートノードに接続できます。詳細は、Outpost に拡張された AWS VPC クラスターでの AWS Classic Load Balancer の使用 を参照してください。
- Outpost インスタンスでロードバランサーを実行するには、AWS Application Load Balancer を使用する必要があります。AWS Load Balancer Operator を使用すると、AWS Load Balancer コントローラーのインスタンスをデプロイできます。このコントローラーは、Kubernetes Ingress リソース用の AWS Application Load Balancer をプロビジョニングします。詳細は、Outpost に拡張された AWS VPC クラスターでの AWS Load Balancer Operator の使用 を参照してください。

20.6.1. Outpost に拡張された AWS VPC クラスターでの AWS Classic Load Balancer の使用

AWS Outposts インフラストラクチャーは、AWS Classic Load Balancer を実行できませんが、AWS VPC クラスターの Classic Load Balancer は、エッジおよびクラウドベースのサブネットが同じアベイラビリティーゾーンにある場合、Outpost のエッジコンピュートノードをターゲットにすることができます。そのため、VPC クラスター上の Classic Load Balancer は、これらのノードタイプのどちらかに Pod をスケジュールする可能性があります。

エッジコンピュートノードおよびクラウドベースのコンピュートノードでワークロードをスケジュールすると、遅延が発生する可能性があります。VPC クラスター内の Classic Load Balancer が Outpost のエッジコンピュートノードをターゲットにするのを防ぐ場合は、クラウドベースのコンピュートノードにラベルを適用し、適用されたラベルを持つノードにのみスケジュールするように Classic Load Balancer を設定できます。

注記

VPC クラスター内の Classic Load Balancer が Outpost エッジコンピュートノードをターゲットにしても問題ない場合、これらの手順を完了する必要はありません。

前提条件

- AWS VPC クラスターを Outpost に拡張した。
- **cluster-admin** 権限を持つアカウントを使用してクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- エッジコンピュートマシンのテイントに一致する容認を持つユーザーワークロードを Outpost に作成している。

手順

1. オプション: 次のコマンドを実行し、出力に Outpost 内のエッジコンピュートノードのみが含まれていることを確認して、エッジコンピュートノードに **location=outposts** ラベルがあることを確認します。

```
$ oc get nodes -l location=outposts
```

2. 次のコマンドを実行して、VPC クラスター内のクラウドベースのコンピュートノードにキーと値のペアのラベルを付けます。

```
$ for NODE in $(oc get node -l node-role.kubernetes.io/worker --no-headers | grep -v outposts | awk '{print$1}'); do oc label node $NODE <key_name>=<value>; done
```

<key_name>=<value> は、クラウドベースのコンピュートノードを識別するために使用するラベルです。

出力例

```
node1.example.com labeled
node2.example.com labeled
node3.example.com labeled
```

3. オプション: 次のコマンドを実行し、出力に VPC クラスター内のすべてのクラウドベースのコンピュートノードが含まれていることを確認して、クラウドベースのコンピュートノードに指定したラベルがあることを確認します。

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

出力例

```
NAME                STATUS   ROLES    AGE     VERSION
node1.example.com   Ready    worker   7h      v1.29.4
node2.example.com   Ready    worker   7h      v1.29.4
node3.example.com   Ready    worker   7h      v1.29.4
```

4. クラウドベースのサブネットの情報を **Service** マニフェストの **annotations** フィールドに追加して、Classic Load Balancer サービスを設定します。

サービス設定の例

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: <application_name>
  name: <application_name>
  namespace: <application_namespace>
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-subnets: <aws_subnet> (1)
    service.beta.kubernetes.io/aws-load-balancer-target-node-labels: <key_name>=<value>
(2)
spec:
  ports:
  - name: http
```

```
    port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    app: <application_name>
  type: LoadBalancer
```

1. AWS VPC クラスターのサブネット ID を指定します。
2. ノードラベルのペアと一致するキーと値のペアを指定します。

5. 次のコマンドを実行して **Service** CR を作成します。

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

検証

1. 次のコマンドを実行して、**service** リソースのステータスを確認して、プロビジョニングされた Classic Load Balancer のホストを表示します。

```
$ HOST=$(oc get service <application_name> -n <application_namespace> --
template='{{(index .status.loadBalancer.ingress 0).hostname}}')
```

2. 次のコマンドを実行して、プロビジョニングされた Classic Load Balancer ホストのステータスを確認します。

```
$ curl $HOST
```

3. AWS コンソールで、ラベル付きインスタンスのみがロードバランサーのターゲットインスタンスとして表示されることを確認します。

20.6.2. Outpost に拡張された AWS VPC クラスターでの AWS Load Balancer Operator の使用

Outpost に拡張された AWS VPC クラスター内で AWS Application Load Balancer をプロビジョニングするように AWS Load Balancer Operator を設定できます。AWS Outposts は AWS Network Load Balancer をサポートしていません。そのため、AWS Load Balancer Operator は Outpost に Network Load Balancer をプロビジョニングできません。

AWS Application Load Balancer は、クラウドサブネットか Outpost サブネットのどちらかに作成できます。クラウドの Application Load Balancer はクラウドベースのコンピュートノードに接続でき、Outpost の Application Load Balancer はエッジコンピュートノードに接続できます。Ingress リソースには Outpost サブネットまたは VPC サブネットのアノテーションを付ける必要がありますが、両方を付けることはできません。

前提条件

- AWS VPC クラスターを Outpost に拡張した。
- OpenShift CLI (**oc**) がインストールされている。
- AWS Load Balancer Operator をインストールし、AWS Load Balancer Controller を作成した。

手順

- 指定のサブネットを使用するように **Ingress** リソースを設定します。

  Ingress リソース設定の例

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <application_name>
  annotations:
    alb.ingress.kubernetes.io/subnets: <subnet_id> 1
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: <application_name>
                port:
                  number: 80
```

  1 使用するサブネットを指定します。

  - Outpost で Application Load Balancer を使用するには、Outpost のサブネット ID を指定します。
  - クラウドで Application Load Balancer を使用するには、別々のアベイラビリティーゾーンに少なくとも 2 つのサブネットを指定する必要があります。

関連情報

- AWS Load Balancer Operator を使用した AWS Load Balancer コントローラーインスタンスの作成

20.7. 関連情報

- AWS のクラスターの既存 VPC へのインストール