# OpenShift Container Platform 4.13

## ホステッドコントロールプレーン

OpenShift Container Platform でホストされたコントロールプレーンを使用する

Last Updated: 2024-04-19

# OpenShift Container Platform 4.13 ホステッドコントロールプレーン

OpenShift Container Platform でホストされたコントロールプレーンを使用する

## 法律上の通知



## 概要

このドキュメントでは、OpenShift Container Platform のホストされたコントロールプレーンを管理するための手順を説明します。ホストされたコントロールプレーンを使用すると、コントロールプレーンごとに専用の物理マシンまたは仮想マシンを必要とせずに、ホストクラスター上にコントロールプレーンを Pod として作成できます。

# 目次

# 第1章 ホストされたコントロールプレーンの概要

OpenShift Container Platform クラスターは、スタンドアロンまたはホストされたコントロールプレーンという 2 つの異なるコントロールプレーン設定を使用してデプロイできます。スタンドアロン設定では、専用の仮想マシンまたは物理マシンを使用してコントロールプレーンをホストします。OpenShift Container Platform のホステッドコントロールプレーンを使用すると、コントロールプレーンごとに専用の仮想または物理マシンを必要とせずに、ホスティングクラスター上にコントロールプレーンを Pod として作成できます。

## 1.1. ホストされたコントロールプレーンの概要 (テクノロジープレビュー)

Red Hat OpenShift Container Platform のホストされたコントロールプレーンを、管理コストの削減、クラスターのデプロイ時間の最適化、管理とワークロードの問題の分離に使用することで、アプリケーションに集中できます。

ホストされたコントロールプレーンをテクノロジープレビュー機能として有効にするには、Amazon Web Services (AWS) 上の Kubernetes Operator バージョン 2.0 以降のマルチクラスターエンジン 、エージェントプロバイダーを使用したベアメタル、または OpenShift Virtualization を使用します。

**重要**

ホストされたコントロールプレーンは、テクノロジープレビュー機能としてのみ利用できます。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場合があります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能では、最新の製品機能をいち早く提供します。これにより、お客様は開発段階で機能をテストし、フィードバックを提供できます。

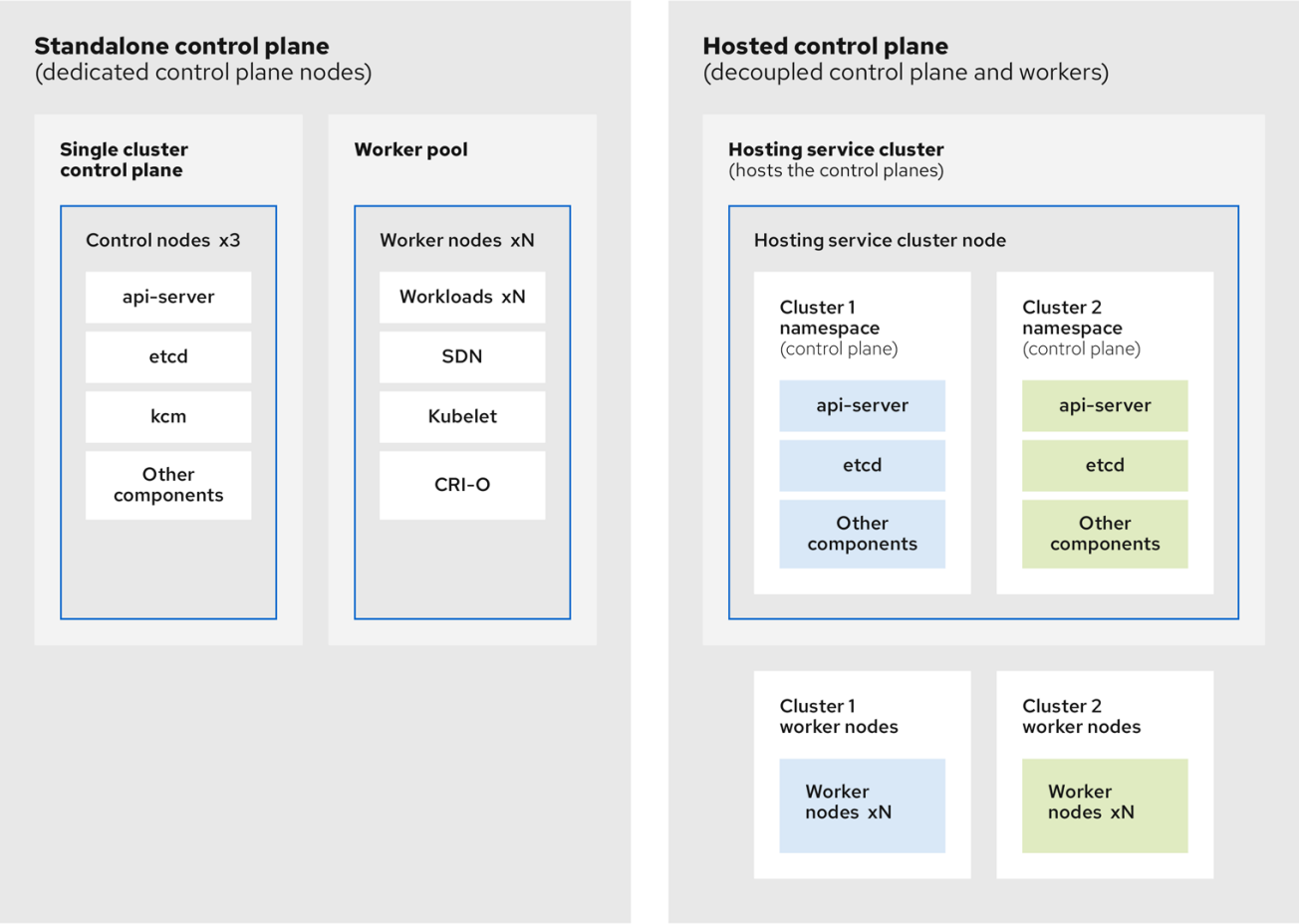
Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

### 1.1.1. ホストされたコントロールプレーンのアーキテクチャー

OpenShift Container Platform は、多くの場合、クラスターがコントロールプレーンとデータプレーンで設定される結合モデルまたはスタンドアロンモデルでデプロイされます。コントロールプレーンには、API エンドポイント、ストレージエンドポイント、ワークロードスケジューラー、および状態を保証するアクチュエーターが含まれます。データプレーンには、ワークロードとアプリケーションが実行されるコンピューティング、ストレージ、ネットワークが含まれます。

スタンドアロンコントロールプレーンは、クォーラムを確保できる最小限の数で、物理または仮想のノードの専用グループによってホストされます。ネットワークスタックは共有されます。クラスターへの管理者アクセスにより、クラスターのコントロールプレーン、マシン管理 API、およびクラスターの状態に影響を与える他のコンポーネントを可視化できます。

スタンドアロンモデルは正常に機能しますが、状況によっては、コントロールプレーンとデータプレーンが分離されたアーキテクチャーが必要になります。そのような場合には、データプレーンは、専用の物理ホスティング環境がある別のネットワークドメインに配置されています。コントロールプレーンは、Kubernetes にネイティブなデプロイやステートフルセットなど、高レベルのプリミティブを使用してホストされます。コントロールプレーンは、他のワークロードと同様に扱われます。




### 1.1.2. ホストされたコントロールプレーンの利点

OpenShift Container Platform のホストされたコントロールプレーンを使用すると、真のハイブリッドクラウドアプローチへの道が開かれ、その他にもいくつかの利点を享受できます。

- コントロールプレーンが分離され、専用のホスティングサービスクラスターでホストされるため、管理とワークロードの間のセキュリティー境界が強化されます。その結果、クラスターのクレデンシャルが他のユーザーに漏洩する可能性が低くなります。インフラストラクチャーのシークレットアカウント管理も分離されているため、クラスターインフラストラクチャーの管理者が誤ってコントロールプレーンインフラストラクチャーを削除することはありません。

- ホストされたコントロールプレーンを使用すると、より少ないノードで多くのコントロールプレーンを実行できます。その結果、クラスターはより安価になります。

- コントロールプレーンは OpenShift Container Platform で起動される Pod で設定されるため、コントロールプレーンはすぐに起動します。同じ原則が、モニタリング、ロギング、自動スケーリングなどのコントロールプレーンとワークロードに適用されます。

- インフラストラクチャーの観点からは、レジストリー、HAProxy、クラスター監視、ストレージノードなどのインフラストラクチャーをテナントのクラウドプロバイダーのアカウントにプッシュして、テナントでの使用を分離できます。

- 運用上の観点からは、マルチクラスター管理はさらに集約され、クラスターの状態と一貫性に影響を与える外部要因が少なくなります。Site Reliability Engineer は、一箇所で問題をデバッグして、クラスターのデータプレインを移動するため、解決までの時間 (TTR) が短縮され、生産性が向上します。

**関連情報**

- HyperShift アドオン (テクノロジープレビュー)
- ホステッドコントロールプレーン (テクノロジープレビュー)

## 1.2. ホストされたコントロールプレーンのバージョン管理

OpenShift Container Platform のメジャー、マイナー、またはパッチバージョンのリリースごとに、ホストされたコントロールプレーンの 2 つのコンポーネントがリリースされます。

- HyperShift Operator
- コマンドラインインターフェイス (CLI)

HyperShift Operator は、**HostedCluster** API リソースによって表されるホストされたクラスターのライフサイクルを管理します。HyperShift Operator は、OpenShift Container Platform の各リリースでリリースされます。HyperShift Operator をインストールすると、次の例に示すように、HyperShift namespace に **supported-versions** という config map が作成されます。config map は、デプロイできる HostedCluster のバージョンを示しています。

```
apiVersion: v1
data:
  supported-versions: '{"versions":["4.13","4.12","4.11"]}'
kind: ConfigMap
metadata:
  labels:
    hypershift.openshift.io/supported-versions: "true"
  name: supported-versions
  namespace: hypershift
```

CLI は、開発目的のヘルパーユーティリティーです。CLI は、HyperShift Operator リリースの一部としてリリースされます。互換性ポリシーは保証されません。

API である **hypershift.openshift.io** は、軽量で柔軟な異種の OpenShift Container Platform クラスターを大規模に作成および管理する方法を提供します。この API は、**HostedCluster** と **NodePool** という 2 つのユーザー向けリソースを公開します。**HostedCluster** リソースは、コントロールプレーンと共通データプレーンの設定をカプセル化します。**HostedCluster** リソースを作成すると、ノードが接続されていない、完全に機能するコントロールプレーンが作成されます。**NodePool** リソースは、**HostedCluster** リソースにアタッチされたスケーラブルなワーカーノードのセットです。

API バージョンポリシーは、通常、Kubernetes API のバージョン管理 のポリシーと一致します。

# 第2章 ホステッドコントロールプレーンの設定

OpenShift Container Platform のホストされたコントロールプレーンの使用を開始するには、まず、使用するプロバイダーでホストされたクラスターを設定します。次に、いくつかの管理タスクを完了します。

**重要**

ホストされたコントロールプレーンは、テクノロジープレビュー機能としてのみ利用できます。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場合があります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能では、最新の製品機能をいち早く提供します。これにより、お客様は開発段階で機能をテストし、フィードバックを提供できます。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、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。

次のプロバイダーのいずれかを選択して、手順を表示できます。

## 2.1. AMAZON WEB SERVICES (AWS)

- AWS でホストクラスターを設定 (テクノロジープレビュー): AWS でホストされたクラスターを設定するタスクには、AWS S3 OIDC シークレットの作成、ルーティング可能なパブリックゾーンの作成、外部 DNS の有効化、AWS PrivateLink の有効化、ホストされたコントロールプレーン機能の有効化、およびホストされたコントロールプレーン CLI のインストールが含まれます。
- AWS でのホストされたコントロールプレーンクラスターの管理 (テクノロジープレビュー): 管理タスクには、AWS でのホストされたクラスターの作成、インポート、アクセス、または削除が含まれます。

## 2.2. ベアメタル

- ベアメタルでのホストクラスターの設定 (テクノロジープレビュー): ホストクラスターを作成する前に DNS を設定します。
- ベアメタル上のホストされたコントロールプレーンクラスターの管理 (テクノロジープレビュー): ホストされたクラスターの作成、**InfraEnv** リソースの作成、エージェントの追加、ホストされたクラスターへのアクセス、**NodePool** オブジェクトのスケーリング、Ingress の処理、ノードの自動スケーリングの有効化、またはホストされたクラスターの削除。

## 2.3. OPENSHIFT VIRTUALIZATION

- OpenShift Virtualization 上でのホストされたコントロールプレーンクラスターの管理 (テクノロジープレビュー): KubeVirt 仮想マシンによってホストされたワーカーノードを使用して OpenShift Container Platform クラスターを作成します。

# 第3章 ホストされたコントロールプレーンの管理

ホステッドコントロールプレーン用に環境を設定し、ホステッドクラスターを作成したら、クラスターとノードをさらに管理できます。

**重要**

ホストされたコントロールプレーンは、テクノロジープレビュー機能としてのみ利用できます。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場合があります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能では、最新の製品機能をいち早く提供します。これにより、お客様は開発段階で機能をテストし、フィードバックを提供できます。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、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。

## 3.1. ホストされたコントロールプレーンの更新

ホストされたコントロールプレーンの更新には、ホストされたクラスターとノードプールの更新が含まれます。更新プロセス中にクラスターが完全に動作し続けるためには、コントロールプレーンとノードの更新を完了する際に、Kubernetes バージョンスキューポリシー の要件を満たす必要があります。

### 3.1.1. ホストされたクラスターの更新

**spec.release** 値は、コントロールプレーンのバージョンを決定します。**HostedCluster** オブジェクトは、意図した **spec.release** 値を **HostedControlPlane.spec.release** 値に送信し、適切なコントロールプレーン Operator バージョンを実行します。

ホストされたコントロールプレーンは、新しいバージョンの Cluster Version Operator (CVO) を通じて、OpenShift Container Platform コンポーネントとともにコントロールプレーンコンポーネントの新しいバージョンのロールアウトを管理します。

### 3.1.2. ノードプールの更新

ノードプールを使用すると、**spec.release** および **spec.config** の値を公開することで、ノードで実行されているソフトウェアを設定できます。次の方法でノードプールのローリング更新を開始できます。

- **spec.release** または **spec.config** の値を変更します。
- AWS インスタンスタイプなどのプラットフォーム固有のフィールドを変更します。結果は、新しいタイプの新規インスタンスのセットになります。
- クラスター設定を変更します (変更がノードに伝播される場合)。

ノードプールは、置換更新とインプレース更新をサポートします。**nodepool.spec.release** 値は、特定のノードプールのバージョンを決定します。**NodePool** オブジェクトは、**.spec.management.upgradeType** 値に従って、置換またはインプレースローリング更新を完了します。

ノードプールを作成した後は、更新タイプは変更できません。更新タイプを変更する場合は、ノードプールを作成し、他のノードプールを削除する必要があります。

#### 3.1.2.1. ノードプールの置き換え更新

**置き換え** 更新では、以前のバージョンから古いインスタンスが削除され、新しいバージョンでインスタンスが作成されます。この更新タイプは、このレベルの不変性がコスト効率に優れているクラウド環境で効果的です。

置き換え更新では、ノードが完全に再プロビジョニングされるため、手動による変更は一切保持されません。

#### 3.1.2.2. ノードプールのインプレース更新

**インプレース** 更新では、インスタンスのオペレーティングシステムが直接更新されます。このタイプは、ベアメタルなど、インフラストラクチャーの制約が高い環境に適しています。

インプレース更新では手動による変更を保存できますが、kubelet 証明書など、クラスターが直接管理するファイルシステムまたはオペレーティングシステムの設定に手動で変更を加えると、エラーが報告されます。

## 3.2. ホストされたコントロールプレーンのノードプールの更新

ホストされたコントロールプレーンでは、ノードプールを更新して OpenShift Container Platform のバージョンを更新します。ノードプールのバージョンは、ホストされているコントロールプレーンのバージョンを超えないものとします。

**手順**

- OpenShift Container Platform の新しいバージョンに更新するプロセスを開始するには、以下のコマンドを入力してノードプールの **spec.release.image** 値を変更します。

```
$ oc -n NAMESPACE patch HC HCNAME --patch '{"spec":{"release":{"image": "example"}}}'
--type=merge
```

**検証**

- 新しいバージョンがロールアウトされたことを確認するには、**.status.version** 値とステータス条件を確認します。

## 3.3. ホストされたコントロールプレーンのノードプールの設定

ホストされたコントロールプレーンでは、管理クラスターの config map 内に **MachineConfig** オブジェクトを作成することでノードプールを設定できます。

**手順**

1. 管理クラスターの config map 内に **MachineConfig** オブジェクトを作成するには、次の情報を入力します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: <configmap-name>
  namespace: clusters
data:
  config: |
    apiVersion: machineconfiguration.openshift.io/v1
    kind: MachineConfig
```

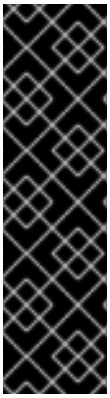
```
  metadata:
    labels:
      machineconfiguration.openshift.io/role: worker
    name: <machineconfig-name>
  spec:
    config:
      ignition:
        version: 3.2.0
      storage:
        files:
        - contents:
            source: data:...
          mode: 420
          overwrite: true
          path: ${PATH} 1
```

1 **MachineConfig** オブジェクトが保存されているノード上のパスを設定します。

2. オブジェクトを config map に追加した後、次のように config map をノードプールに適用できます。

```
spec:
  config:
    - name: ${CONFIGMAP_NAME}
```

## 3.4. ホストされたクラスターにおけるノードのチューニング設定



**重要**

ホストされたコントロールプレーンは、テクノロジープレビュー機能としてのみ利用できます。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場合があります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能では、最新の製品機能をいち早く提供します。これにより、お客様は開発段階で機能をテストし、フィードバックを提供できます。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、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。

ホストされたクラスター内のノードでノードレベルのチューニングを設定するには、Node Tuning Operator を使用できます。ホストされたコントロールプレーンでは、**Tuned** オブジェクトを含む設定マップを作成し、ノードプールでそれらの設定マップを参照することで、ノードのチューニングを設定できます。

**手順**

1. チューニングされた有効なマニフェストを含む設定マップを作成し、ノードプールでマニフェストを参照します。次の例で **Tuned** マニフェストは、任意の値を持つ **tuned-1-node-label** ノードラベルを含むノード上で **vm.dirty_ratio** を 55 に設定するプロファイルを定義します。次の **ConfigMap** マニフェストを **tuned-1.yaml** という名前のファイルに保存します。

```
  apiVersion: v1
  kind: ConfigMap
```

```
metadata:
  name: tuned-1
  namespace: clusters
data:
  tuning: |
    apiVersion: tuned.openshift.io/v1
    kind: Tuned
    metadata:
      name: tuned-1
      namespace: openshift-cluster-node-tuning-operator
    spec:
      profile:
      - data: |
          [main]
          summary=Custom OpenShift profile
          include=openshift-node
          [sysctl]
          vm.dirty_ratio="55"
        name: tuned-1-profile
      recommend:
      - priority: 20
        profile: tuned-1-profile
```

**注記**

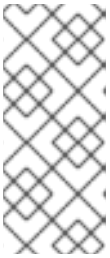
Tuned 仕様の **spec.recommend** セクションのエントリーにラベルを追加しない場合は、ノードプールベースのマッチングが想定されるため、**spec.recommend** セクションの最も優先度の高いプロファイルがプール内のノードに適用されます。Tuned **.spec.recommend.match** セクションでラベル値を設定することにより、よりきめ細かいノードラベルベースのマッチングを実現できますが、ノードプールの **.spec.management.upgradeType** 値を **InPlace** に 設定しない限り、ノードラベルはアップグレード中に保持されません。

2. 管理クラスターに **ConfigMap** オブジェクトを作成します。

```
$ oc --kubeconfig="$MGMT_KUBECONFIG" create -f tuned-1.yaml
```

3. ノードプールを編集するか作成して、ノードプールの **spec.tuningConfig** フィールドで **ConfigMap** オブジェクトを参照します。この例では、2 つのノードを含む **nodepool-1** という名前の **NodePool** が 1 つだけあることを前提としています。

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
  ...
  name: nodepool-1
  namespace: clusters
...
spec:
  ...
  tuningConfig:
  - name: tuned-1
status:
...
```



**注記**

複数のノードプールで同じ設定マップを参照できます。ホストされたコントロールプレーンでは、Node Tuning Operator はノードプール名と namespace のハッシュを Tuned CR の名前に追加してそれらを区別します。このケース以外では、同じホストクラスターの異なる Tuned CR に同じ名前の複数の Tuned プロファイルを作成しないでください。

**検証**

これで **Tuned** マニフェストを含む **ConfigMap** オブジェクトを作成し、それを **NodePool** で参照しました。次に、Node Tuning Operator は **Tuned** オブジェクトをホストされたクラスターに同期します。どの **Tuned** オブジェクトが定義されているか、どの Tuned プロファイルが各ノードに適用されているかを確認できます。

1. ホストされたクラスター内の **Tuned** オブジェクトを一覧表示します。

```
$ oc --kubeconfig="$HC_KUBECONFIG" get tuned.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

**出力例**

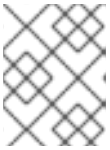
```
NAME       AGE
default    7m36s
rendered   7m36s
tuned-1    65s
```

2. ホストされたクラスター内の **Profile** オブジェクトを一覧表示します。

```
$ oc --kubeconfig="$HC_KUBECONFIG" get profile.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

**出力例**

```
NAME                   TUNED          APPLIED   DEGRADED   AGE
nodepool-1-worker-1    tuned-1-profile  True      False      7m43s
nodepool-1-worker-2    tuned-1-profile  True      False      7m14s
```



**注記**

カスタムプロファイルが作成されていない場合は、**openshift-node** プロファイルがデフォルトで適用されます。

3. チューニングが正しく適用されたことを確認するには、ノードでデバッグシェルを開始し、sysctl 値を確認します。

```
$ oc --kubeconfig="$HC_KUBECONFIG" debug node/nodepool-1-worker-1 -- chroot /host sysctl vm.dirty_ratio
```

**出力例**

```
vm.dirty_ratio = 55
```

## 3.5. ホストされたコントロールプレーン用の SR-IOV OPERATOR のデプロイ

**重要**

ホストされたコントロールプレーンは、テクノロジープレビュー機能としてのみ利用できます。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場合があります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能では、最新の製品機能をいち早く提供します。これにより、お客様は開発段階で機能をテストし、フィードバックを提供できます。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、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。

ホスティングサービスクラスターを設定してデプロイすると、ホストされたクラスターで SR-IOV Operator へのサブスクリプションを作成できます。SR-IOV Pod は、コントロールプレーンではなくワーカーマシンで実行されます。

### 前提条件

AWS 上でホストされたクラスターを設定およびデプロイしている。詳細は、AWS 上でのホストクラスターの設定 (テクノロジープレビュー) を参照してください。

### 手順

1. namespace と Operator グループを作成します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
  - openshift-sriov-network-operator
```

2. SR-IOV Operator へのサブスクリプションを作成します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subsription
  namespace: openshift-sriov-network-operator
spec:
  channel: "4.13"
  name: sriov-network-operator
  config:
    nodeSelector:
```

```
    node-role.kubernetes.io/worker: ""
  source: s/qe-app-registry/redhat-operators
  sourceNamespace: openshift-marketplace
```

**検証**

1. SR-IOV Operator の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、結果の出力を表示します。

```
$ oc get csv -n openshift-sriov-network-operator
```

**出力例**

```
NAME                                         DISPLAY                   VERSION               REPLACES
PHASE
sriov-network-operator.4.13.0-202211021237   SR-IOV Network Operator   4.13.0-
202211021237   sriov-network-operator.4.13.0-202210290517   Succeeded
```

2. SR-IOV Pod がデプロイ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 pods -n openshift-sriov-network-operator
```

## 3.6. ホストされたクラスターの削除

ホストされたクラスターを削除する手順は、使用するプロバイダーによって異なります。

**手順**

- クラスターが AWS にある場合は、AWS でのホストされたクラスターの破棄 の手順に従います。
- クラスターがベアメタル上にある場合は、ベアメタルでのホストされたクラスターの破棄 の手順に従います。
- クラスターが OpenShift Virtualization にある場合は、OpenShift Virtualization 上のホストされたクラスターの破棄 の手順に従います。

**次のステップ**

ホステッドコントロールプレーン機能を無効にする場合は、ホストされたコントロールプレーン機能の無効化 を参照してください。

# 第4章 ホストされたコントロールプレーンのバックアップ、復元、障害復旧

ホストされたクラスターで etcd をバックアップおよび復元する必要がある場合、またはホストされたクラスターの障害復旧を行う必要がある場合は、次の手順を確認してください。

**重要**

ホストされたコントロールプレーンは、テクノロジープレビュー機能としてのみ利用できます。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場合があります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能では、最新の製品機能をいち早く提供します。これにより、お客様は開発段階で機能をテストし、フィードバックを提供できます。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、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。

## 4.1. ホストされたクラスターでの ETCD のバックアップと復元

OpenShift Container Platform でホストされたコントロールプレーンを使用する場合、etcd をバックアップおよび復元するプロセスは、通常の etcd バックアッププロセス とは異なります。

### 4.1.1. ホストされたクラスターでの etcd のスナップショットの作成

ホストされたクラスターの etcd をバックアップするプロセスの一環として、etcd のスナップショットを作成します。スナップショットを作成した後、たとえば障害復旧操作の一部として復元できます。

**重要**

この手順には、API のダウンタイムが必要です。

**手順**

1. このコマンドを入力して、ホストされたクラスターの調整を一時停止します。

```
$ oc patch -n clusters hostedclusters/${CLUSTER_NAME} -p '{"spec":
{"pausedUntil":"${PAUSED_UNTIL}"}}' --type=merge
```

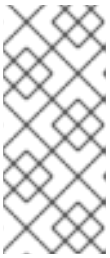
2. このコマンドを入力して、すべての etcd-writer デプロイメントを停止します。

```
$ oc scale deployment -n ${HOSTED_CLUSTER_NAMESPACE} --replicas=0 kube-
apiserver openshift-apiserver openshift-oauth-apiserver
```

3. 各 etcd コンテナーで **exec** コマンドを使用して、etcd スナップショットを作成します。

```
$ oc exec -it etcd-0 -n ${HOSTED_CLUSTER_NAMESPACE} -- env ETCDCTL_API=3
/usr/bin/etcdctl --cacert /etc/etcd/tls/client/etcd-client-ca.crt --cert /etc/etcd/tls/client/etcd-
client.crt --key /etc/etcd/tls/client/etcd-client.key --endpoints=localhost:2379 snapshot save
/var/lib/data/snapshot.db
$ oc exec -it etcd-0 -n ${HOSTED_CLUSTER_NAMESPACE} -- env ETCDCTL_API=3
/usr/bin/etcdctl -w table snapshot status /var/lib/data/snapshot.db
```

4. 次の例に示すように、S3 バケットなど、後で取得できる場所にスナップショットデータをコピーします。



**注記**

次の例では、署名バージョン 2 を使用しています。署名バージョン 4 をサポートするリージョン (例: us-east-2 リージョン) にいる場合は、署名バージョン 4 を使用してください。それ以外の場合は、署名バージョン 2 を使用してスナップショットを S3 バケットにコピーすると、アップロードは失敗し、署名バージョン 2 は非推奨になります。

**例**

```
BUCKET_NAME=somebucket
FILEPATH="/${BUCKET_NAME}/${CLUSTER_NAME}-snapshot.db"
CONTENT_TYPE="application/x-compressed-tar"
DATE_VALUE=`date -R`
SIGNATURE_STRING="PUT\n\n${CONTENT_TYPE}\n${DATE_VALUE}\n${FILEPATH}"
ACCESS_KEY=accesskey
SECRET_KEY=secret
SIGNATURE_HASH=`echo -en ${SIGNATURE_STRING} | openssl sha1 -hmac
${SECRET_KEY} -binary | base64`

oc exec -it etcd-0 -n ${HOSTED_CLUSTER_NAMESPACE} -- curl -X PUT -T
"/var/lib/data/snapshot.db" \
  -H "Host: ${BUCKET_NAME}.s3.amazonaws.com" \
  -H "Date: ${DATE_VALUE}" \
  -H "Content-Type: ${CONTENT_TYPE}" \
  -H "Authorization: AWS ${ACCESS_KEY}:${SIGNATURE_HASH}" \
  https://${BUCKET_NAME}.s3.amazonaws.com/${CLUSTER_NAME}-snapshot.db
```

5. 後で新しいクラスターでスナップショットを復元できるようにする場合は、この例に示すように、ホストされたクラスターが参照する暗号化シークレットを保存します。

**例**

```
oc get hostedcluster $CLUSTER_NAME -o=jsonpath='{.spec.secretEncryption.aescbc}'
{"activeKey":{"name":"CLUSTER_NAME-etcd-encryption-key"}}

# Save this secret, or the key it contains so the etcd data can later be decrypted
oc get secret ${CLUSTER_NAME}-etcd-encryption-key -o=jsonpath='{.data.key}'
```

### 次のステップ

etcd スナップショットを復元します。

## 4.1.2. ホストされたクラスターでの etcd スナップショットの復元

ホストされたクラスターからの etcd のスナップショットがある場合は、それを復元できます。現在、クラスターの作成中にのみ etcd スナップショットを復元できます。

etcd スナップショットを復元するには、**create cluster --render** コマンドからの出力を変更し、**HostedCluster** 仕様の etcd セクションで **restoreSnapshotURL** 値を定義します。

### 前提条件

ホストされたクラスターで etcd スナップショットを作成している。

### 手順

1. **aws** コマンドラインインターフェイス (CLI) で事前に署名された URL を作成し、認証情報を etcd デプロイメントに渡さずに S3 から etcd スナップショットをダウンロードできるようにします。

```
ETCD_SNAPSHOT=${ETCD_SNAPSHOT:-"s3://${BUCKET_NAME}/${CLUSTER_NAME}-snapshot.db"}
ETCD_SNAPSHOT_URL=$(aws s3 presign ${ETCD_SNAPSHOT})
```

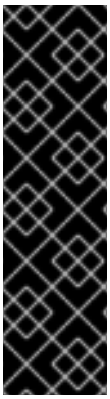
2. 次の URL を参照するように **HostedCluster** 仕様を変更します。

```
spec:
  etcd:
    managed:
      storage:
        persistentVolume:
          size: 4Gi
        type: PersistentVolume
        restoreSnapshotURL:
        - "${ETCD_SNAPSHOT_URL}"
    managementType: Managed
```

3. **spec.secretEncryption.aescbc** 値から参照したシークレットに、前の手順で保存したものと同じ AES キーが含まれていることを確認します。

## 4.2. AWS リージョン内のホストされたクラスターの障害復旧

ホストされたクラスターの障害復旧 (DR) が必要な状況では、ホストされたクラスターを AWS 内の同じリージョンに回復できます。たとえば、管理クラスターのアップグレードが失敗し、ホストされているクラスターが読み取り専用状態になっている場合、DR が必要です。



**重要**

ホストされたコントロールプレーンは、テクノロジープレビュー機能としてのみ利用できます。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場合があります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能では、最新の製品機能をいち早く提供します。これにより、お客様は開発段階で機能をテストし、フィードバックを提供できます。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、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。

DR プロセスには、次の 3 つの主要な手順が含まれます。

1. ソース管理クラスターでのホストされたクラスターのバックアップ
2. 宛先管理クラスターでのホストされたクラスターの復元

3. ホストされたクラスターのソース管理クラスターからの削除

プロセス中、ワークロードは引き続き実行されます。クラスター API は一定期間使用できない場合がありますが、ワーカーノードで実行されているサービスには影響しません。

**重要**

次の例に示すように、ソース管理クラスターと宛先管理クラスターの両方に **--external-dns** フラグを設定して、API サーバー URL を維持する必要があります。

**例: 外部 DNS フラグ**

```
--external-dns-provider=aws \
--external-dns-credentials=<AWS Credentials location> \
--external-dns-domain-filter=<DNS Base Domain>
```

これにより、サーバー URL は **https://api-sample-hosted.sample-hosted.aws.openshift.com** で終わります。

API サーバー URL を維持するために **--external-dns** フラグを含めない場合、ホストされたクラスターを移行することはできません。

### 4.2.1. 環境とコンテキストの例

復元するクラスターが 3 つあるシナリオを考えてみます。2 つは管理クラスターで、1 つはホストされたクラスターです。コントロールプレーンのみ、またはコントロールプレーンとノードのいずれかを復元できます。開始する前に、以下の情報が必要です。

- ソース MGMT namespace: ソース管理 namespace
- ソース MGMT ClusterName: ソース管理クラスター名
- ソース MGMT Kubeconfig: ソース管理 **kubeconfig** ファイル
- 宛先 MGMT Kubeconfig: 宛先管理 **kubeconfig** ファイル
- HC Kubeconfig: ホストされたクラスターの **kubeconfig** ファイル
- SSH 鍵ファイル: SSH 公開鍵
- プルシークレット: リリースイメージにアクセスするためのプルシークレットファイル
- AWS 認証情報
- AWS region
- ベースドメイン: 外部 DNS として使用する DNS ベースドメイン
- S3 バケット名: etcd バックアップをアップロードする予定の AWS リージョンのバケット

この情報は、次の環境変数の例に示されています。

**環境変数の例**

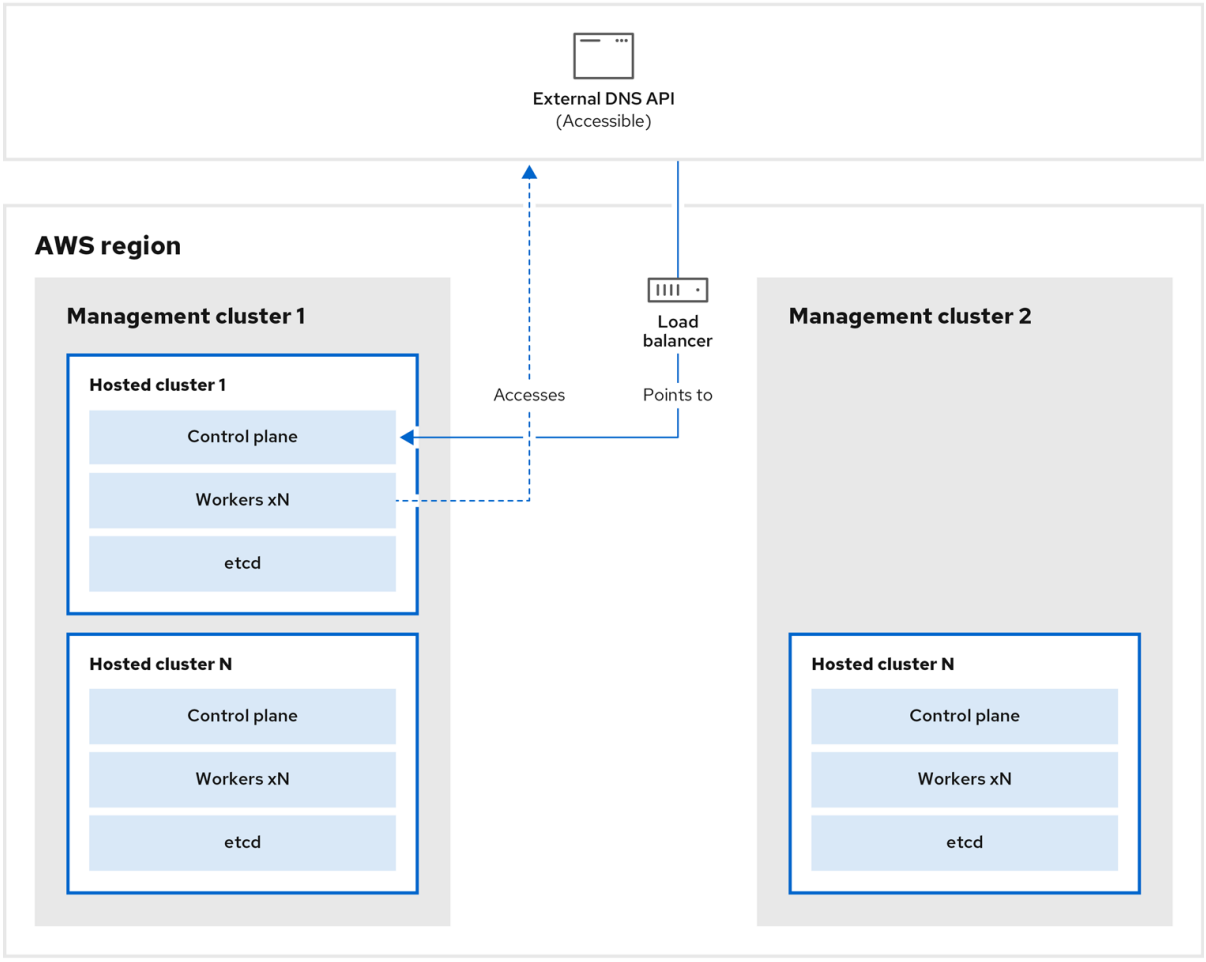
```
SSH_KEY_FILE=${HOME}/.ssh/id_rsa.pub
BASE_PATH=${HOME}/hypershift
```

```
BASE_DOMAIN="aws.sample.com"
PULL_SECRET_FILE="${HOME}/pull_secret.json"
AWS_CREDS="${HOME}/.aws/credentials"
AWS_ZONE_ID="Z02718293M33QHDEQBROL"

CONTROL_PLANE_AVAILABILITY_POLICY=SingleReplica
HYPERSHIFT_PATH=${BASE_PATH}/src/hypershift
HYPERSHIFT_CLI=${HYPERSHIFT_PATH}/bin/hypershift
HYPERSHIFT_IMAGE=${HYPERSHIFT_IMAGE:-"quay.io/${USER}/hypershift:latest"}
NODE_POOL_REPLICAS=${NODE_POOL_REPLICAS:-2}

# MGMT Context
MGMT_REGION=us-west-1
MGMT_CLUSTER_NAME="${USER}-dev"
MGMT_CLUSTER_NS=${USER}
MGMT_CLUSTER_DIR="${BASE_PATH}/hosted_clusters/${MGMT_CLUSTER_NS}-
${MGMT_CLUSTER_NAME}"
MGMT_KUBECONFIG="${MGMT_CLUSTER_DIR}/kubeconfig"

# MGMT2 Context
MGMT2_CLUSTER_NAME="${USER}-dest"
MGMT2_CLUSTER_NS=${USER}
MGMT2_CLUSTER_DIR="${BASE_PATH}/hosted_clusters/${MGMT2_CLUSTER_NS}-
${MGMT2_CLUSTER_NAME}"
MGMT2_KUBECONFIG="${MGMT2_CLUSTER_DIR}/kubeconfig"

# Hosted Cluster Context
HC_CLUSTER_NS=clusters
HC_REGION=us-west-1
HC_CLUSTER_NAME="${USER}-hosted"
HC_CLUSTER_DIR="${BASE_PATH}/hosted_clusters/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}"
HC_KUBECONFIG="${HC_CLUSTER_DIR}/kubeconfig"
BACKUP_DIR=${HC_CLUSTER_DIR}/backup

BUCKET_NAME="${USER}-hosted-${MGMT_REGION}"

# DNS
AWS_ZONE_ID="Z07342811SH9AA102K1AC"
EXTERNAL_DNS_DOMAIN="hc.jpdv.aws.kerbeross.com"
```

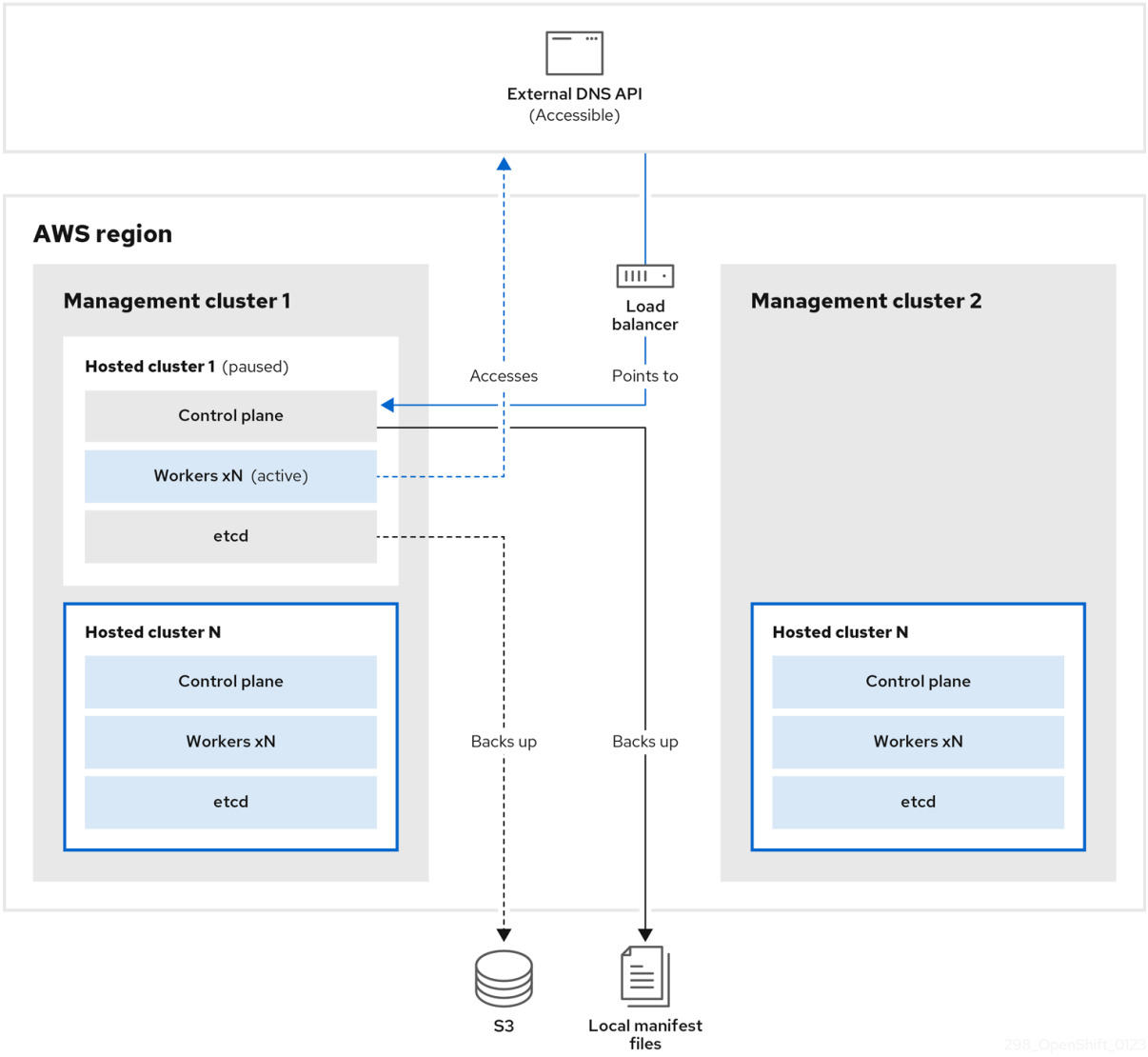
### 4.2.2. バックアップおよび復元プロセスの概要

バックアップと復元のプロセスは、以下のような仕組みとなっています。

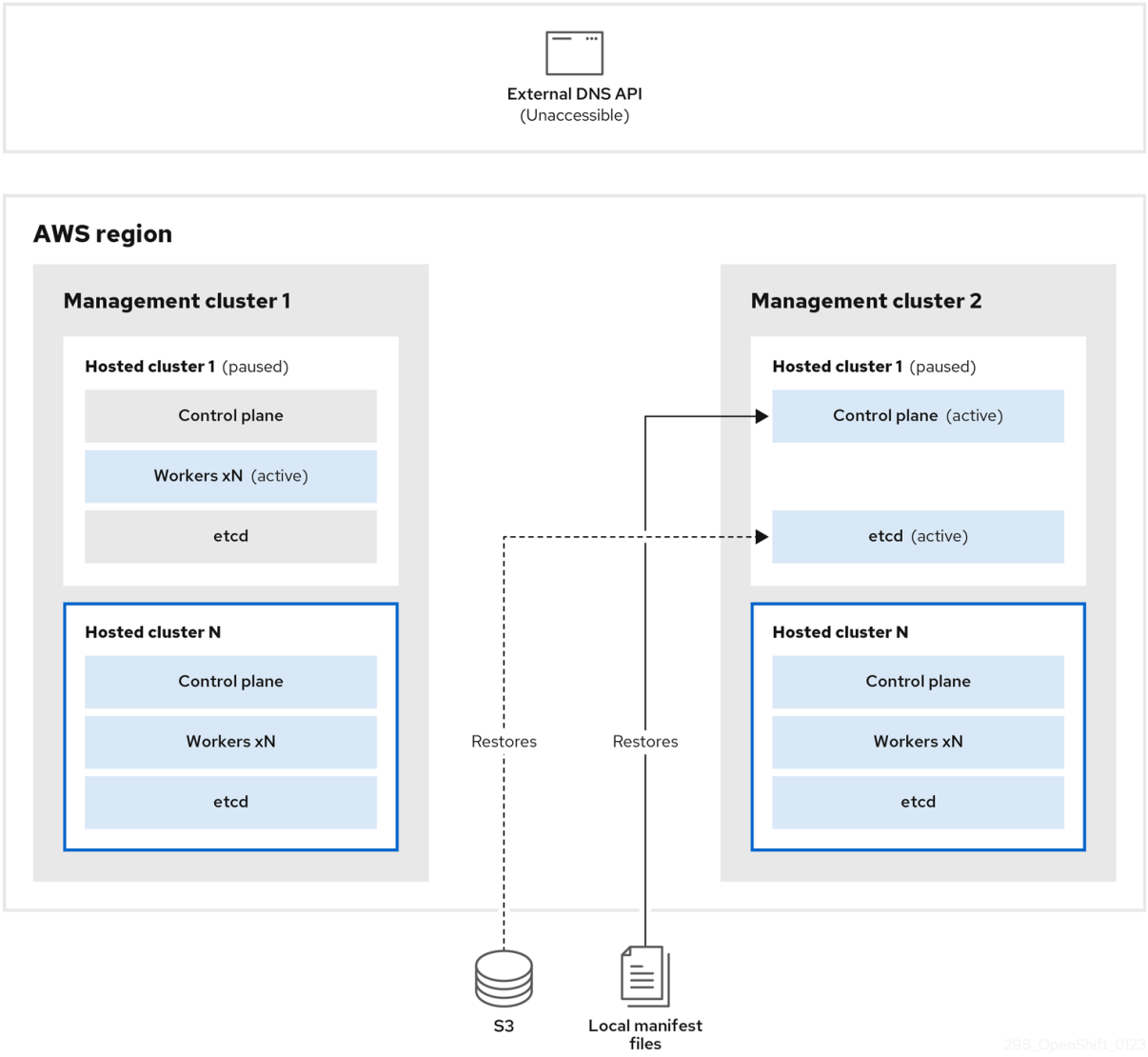
1. 管理クラスター 1 (ソース管理クラスターと見なすことができます) では、コントロールプレーンとワーカーが 外部 DNS API を使用して対話します。外部 DNS API はアクセス可能で、管理クラスター間にロードバランサーが配置されています。



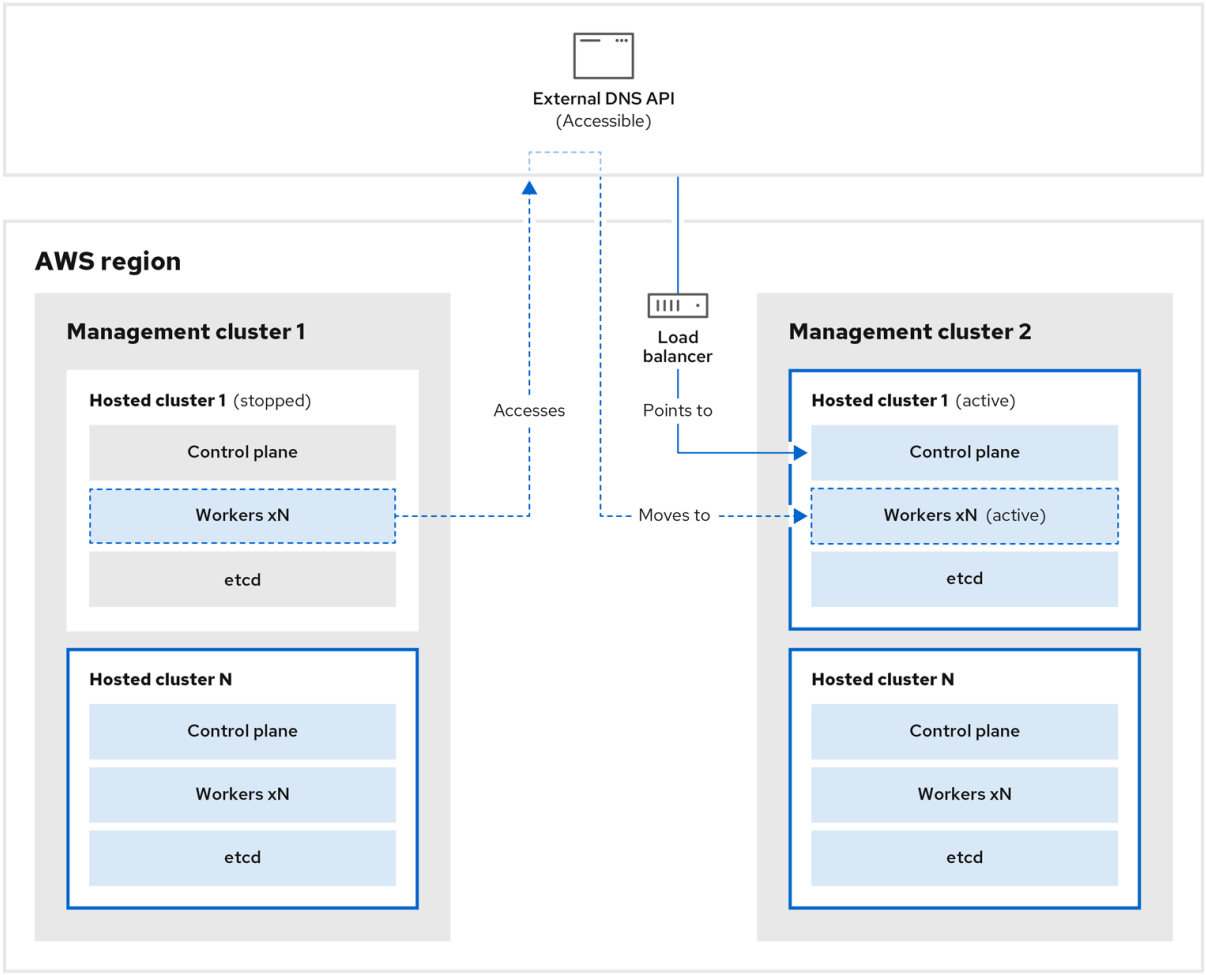

2. ホストされたクラスターのスナップショットを作成します。これには、etcd、コントロールプレーン、およびワーカーノードが含まれます。このプロセスの間、ワーカーノードは外部 DNS API にアクセスできなくても引き続きアクセスを試みます。また、ワークロードが実行され、コントロールプレーンがローカルマニフェストファイルに保存され、etcd が S3 バケットにバックアップされます。データプレーンはアクティブで、コントロールプレーンは一時停止しています。



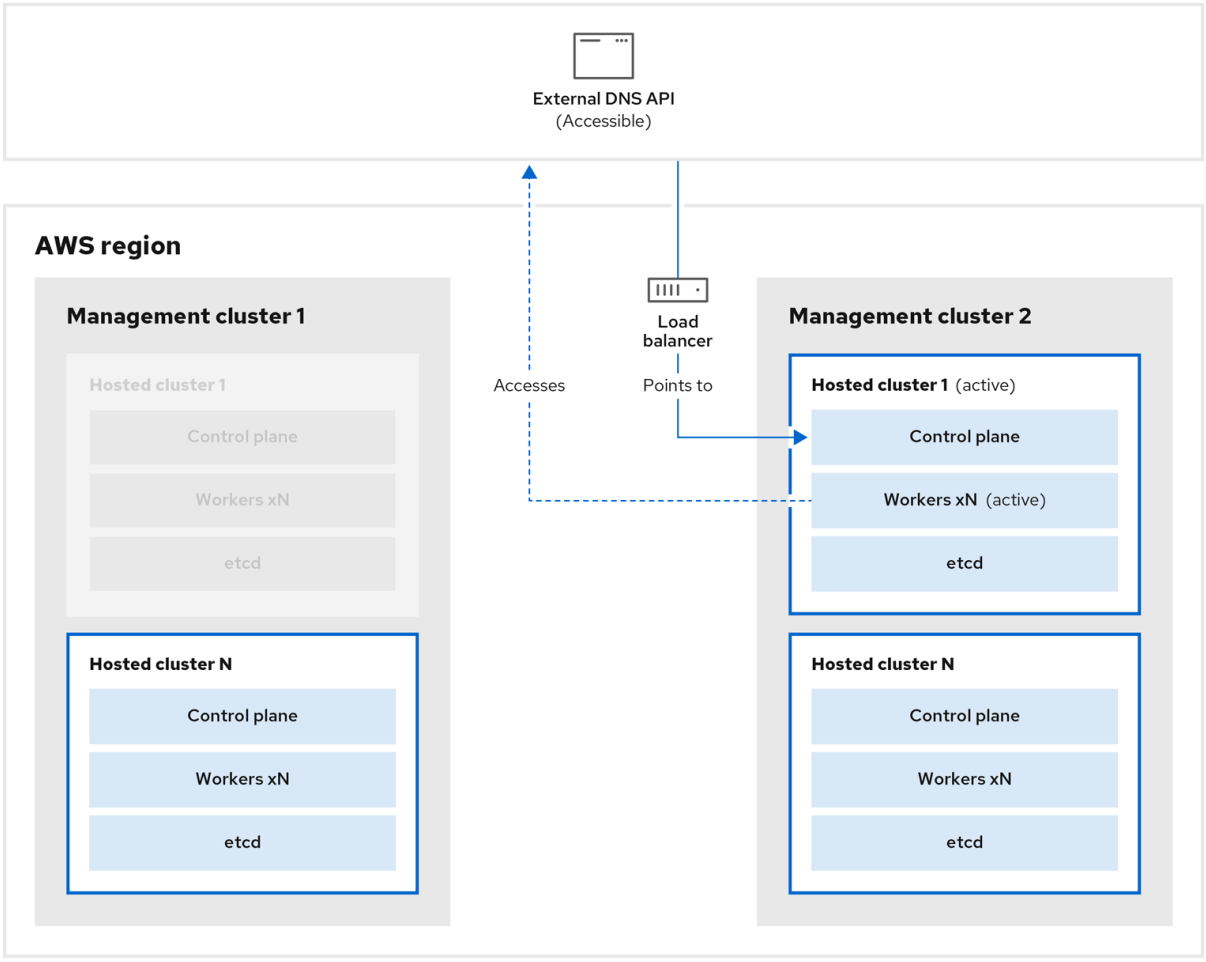

3. 管理クラスター 2 (宛先管理クラスターと見なすことができます) では、S3 バケットから etcd を復元し、ローカルマニフェストファイルからコントロールプレーンを復元します。このプロセスの間、外部 DNS API は停止し、ホストされたクラスター API にアクセスできなくなり、API を使用するワーカーはマニフェストファイルを更新できなくなりますが、ワークロードは引き続き実行されます。




4. 外部 DNS API に再びアクセスできるようになり、ワーカーノードはそれを使用して管理クラスター 2 に移動します。外部 DNS API は、コントロールプレーンを参照するロードバランサーにアクセスできます。




5. 管理クラスター 2 では、コントロールプレーンとワーカーノードが外部 DNS API を使用して対話します。リソースは、etcd の S3 バックアップを除いて、管理クラスター 1 から削除されます。ホストされたクラスターを管理クラスター 1 で再度設定しようとしても、機能しません。




ホストされたクラスターを手動でバックアップおよび復元するか、スクリプトを実行してプロセスを完了することができます。スクリプトの詳細については、「ホストされたクラスターをバックアップおよび復元するためのスクリプトの実行」を参照してください。

### 4.2.3. ホストされたクラスターのバックアップ

ターゲット管理クラスターでホストされたクラスターを復元するには、最初にすべての関連データをバックアップする必要があります。

**手順**

1. 以下のコマンドを入力して、configmap ファイルを作成し、ソース管理クラスターを宣言します。

```
$ oc create configmap mgmt-parent-cluster -n default --from-
literal=from=${MGMT_CLUSTER_NAME}
```

2. 以下のコマンドを入力して、ホストされたクラスターとノードプールの調整をシャットダウンします。

```
PAUSED_UNTIL="true"
oc patch -n ${HC_CLUSTER_NS} hostedclusters/${HC_CLUSTER_NAME} -p '{"spec":
{"pausedUntil":"${PAUSED_UNTIL}"}}' --type=merge
oc scale deployment -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --replicas=0 kube-
apiserver openshift-apiserver openshift-oauth-apiserver control-plane-operator
```

–

```
PAUSED_UNTIL="true"
oc patch -n ${HC_CLUSTER_NS} hostedclusters/${HC_CLUSTER_NAME} -p '{"spec":
{"pausedUntil":"${PAUSED_UNTIL}"}}' --type=merge
oc patch -n ${HC_CLUSTER_NS} nodepools/${NODEPOOLS} -p '{"spec":
{"pausedUntil":"${PAUSED_UNTIL}"}}' --type=merge
oc scale deployment -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --replicas=0 kube-
apiserver openshift-apiserver openshift-oauth-apiserver control-plane-operator
```

3. 以下の bash スクリプトを実行して、etcd をバックアップし、データを S3 バケットにアップロードします。

**ヒント**

このスクリプトを関数でラップし、メイン関数から呼び出します。

```
# ETCD Backup
ETCD_PODS="etcd-0"
if [ "${CONTROL_PLANE_AVAILABILITY_POLICY}" = "HighlyAvailable" ]; then
  ETCD_PODS="etcd-0 etcd-1 etcd-2"
fi

for POD in ${ETCD_PODS}; do
  # Create an etcd snapshot
  oc exec -it ${POD} -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -- env
ETCDCTL_API=3 /usr/bin/etcdctl --cacert /etc/etcd/tls/client/etcd-client-ca.crt --cert
/etc/etcd/tls/client/etcd-client.crt --key /etc/etcd/tls/client/etcd-client.key --
endpoints=localhost:2379 snapshot save /var/lib/data/snapshot.db
  oc exec -it ${POD} -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -- env
ETCDCTL_API=3 /usr/bin/etcdctl -w table snapshot status /var/lib/data/snapshot.db

  FILEPATH="/${BUCKET_NAME}/${HC_CLUSTER_NAME}-${POD}-snapshot.db"
  CONTENT_TYPE="application/x-compressed-tar"
  DATE_VALUE=`date -R`
  SIGNATURE_STRING="PUT\n\n${CONTENT_TYPE}\n${DATE_VALUE}\n${FILEPATH}"

  set +x
  ACCESS_KEY=$(grep aws_access_key_id ${AWS_CREDS} | head -n1 | cut -d= -f2 | sed
"s/ //g")
  SECRET_KEY=$(grep aws_secret_access_key ${AWS_CREDS} | head -n1 | cut -d= -f2 |
sed "s/ //g")
  SIGNATURE_HASH=$(echo -en ${SIGNATURE_STRING} | openssl sha1 -hmac
"${SECRET_KEY}" -binary | base64)
  set -x

  # FIXME: this is pushing to the OIDC bucket
  oc exec -it etcd-0 -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -- curl -X PUT -T
"/var/lib/data/snapshot.db" \
    -H "Host: ${BUCKET_NAME}.s3.amazonaws.com" \
    -H "Date: ${DATE_VALUE}" \
    -H "Content-Type: ${CONTENT_TYPE}" \
    -H "Authorization: AWS ${ACCESS_KEY}:${SIGNATURE_HASH}" \
    https://${BUCKET_NAME}.s3.amazonaws.com/${HC_CLUSTER_NAME}-${POD}-
snapshot.db
done
```

etcd のバックアップの詳細については、「ホストされたクラスターでの etcd のバックアップと復元」を参照してください。

4. 以下のコマンドを入力して、Kubernetes および OpenShift Container Platform オブジェクトをバックアップします。次のオブジェクトをバックアップする必要があります。

   - HostedCluster namespace の **HostedCluster** および **NodePool** オブジェクト
   - HostedCluster namespace の **HostedCluster** シークレット
   - Hosted Control Plane namespace の **HostedControlPlane**
   - Hosted Control Plane namespace の **Cluster**
   - Hosted Control Plane namespace の **AWSCluster**、**AWSMachineTemplate**、および **AWSMachine**
   - Hosted Control Plane namespace の **MachineDeployments**、**MachineSets**、および **Machines**
   - Hosted Control Plane namespace の **ControlPlane** シークレット

```
mkdir -p ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}
chmod 700 ${BACKUP_DIR}/namespaces/

# HostedCluster
echo "Backing Up HostedCluster Objects:"
oc get hc ${HC_CLUSTER_NAME} -n ${HC_CLUSTER_NS} -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/hc-${HC_CLUSTER_NAME}.yaml
echo "--> HostedCluster"
sed -i '' -e '/^status:$/,$d' ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/hc-
${HC_CLUSTER_NAME}.yaml

# NodePool
oc get np ${NODEPOOLS} -n ${HC_CLUSTER_NS} -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/np-${NODEPOOLS}.yaml
echo "--> NodePool"
sed -i '' -e '/^status:$/,$ d' ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/np-
${NODEPOOLS}.yaml

# Secrets in the HC Namespace
echo "--> HostedCluster Secrets:"
for s in $(oc get secret -n ${HC_CLUSTER_NS} | grep "^${HC_CLUSTER_NAME}" |
awk '{print $1}'); do
    oc get secret -n ${HC_CLUSTER_NS} $s -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/secret-${s}.yaml
done

# Secrets in the HC Control Plane Namespace
echo "--> HostedCluster ControlPlane Secrets:"
for s in $(oc get secret -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} | egrep -v
"docker|service-account-token|oauth-openshift|NAME|token-${HC_CLUSTER_NAME}" |
awk '{print $1}'); do
    oc get secret -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} $s -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}/secret-
```

```
${s}.yaml
done

# Hosted Control Plane
echo "--> HostedControlPlane:"
oc get hcp ${HC_CLUSTER_NAME} -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}
-o yaml > ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/hcp-${HC_CLUSTER_NAME}.yaml

# Cluster
echo "--> Cluster:"
CL_NAME=$(oc get hcp ${HC_CLUSTER_NAME} -n ${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME} -o jsonpath={.metadata.labels.\*} | grep
${HC_CLUSTER_NAME})
oc get cluster ${CL_NAME} -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -o yaml
> ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}/cl-
${HC_CLUSTER_NAME}.yaml

# AWS Cluster
echo "--> AWS Cluster:"
oc get awscluster ${HC_CLUSTER_NAME} -n ${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME} -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}/awscl-
${HC_CLUSTER_NAME}.yaml

# AWS MachineTemplate
echo "--> AWS Machine Template:"
oc get awsmachinetemplate ${NODEPOOLS} -n ${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME} -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}/awsmt-
${HC_CLUSTER_NAME}.yaml

# AWS Machines
echo "--> AWS Machine:"
CL_NAME=$(oc get hcp ${HC_CLUSTER_NAME} -n ${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME} -o jsonpath={.metadata.labels.\*} | grep
${HC_CLUSTER_NAME})
for s in $(oc get awsmachines -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --no-
headers | grep ${CL_NAME} | cut -f1 -d\ ); do
   oc get -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} awsmachines $s -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}/awsm-
${s}.yaml
done

# MachineDeployments
echo "--> HostedCluster MachineDeployments:"
for s in $(oc get machinedeployment -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}
-o name); do
   mdp_name=$(echo ${s} | cut -f 2 -d /)
   oc get -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} $s -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machinedeployment-${mdp_name}.yaml
done

# MachineSets
echo "--> HostedCluster MachineSets:"
```

```
for s in $(oc get machineset -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -o
name); do
    ms_name=$(echo ${s} | cut -f 2 -d /)
    oc get -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} $s -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machineset-${ms_name}.yaml
done

# Machines
echo "--> HostedCluster Machine:"
for s in $(oc get machine -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -o name);
do
    m_name=$(echo ${s} | cut -f 2 -d /)
    oc get -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} $s -o yaml >
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machine-${m_name}.yaml
done
```

5. 次のコマンドを入力して、**ControlPlane** ルートをクリーンアップします。

```
$ oc delete routes -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --all
```

このコマンドを入力すると、ExternalDNS Operator が Route53 エントリーを削除できるようになります。

6. 次のスクリプトを実行して、Route53 エントリーがクリーンであることを確認します。

```
function clean_routes() {

    if [[ -z "${1}" ]];then
        echo "Give me the NS where to clean the routes"
        exit 1
    fi

    # Constants
    if [[ -z "${2}" ]];then
        echo "Give me the Route53 zone ID"
        exit 1
    fi

    ZONE_ID=${2}
    ROUTES=10
    timeout=40
    count=0

    # This allows us to remove the ownership in the AWS for the API route
    oc delete route -n ${1} --all

    while [ ${ROUTES} -gt 2 ]
    do
        echo "Waiting for ExternalDNS Operator to clean the DNS Records in AWS Route53
where the zone id is: ${ZONE_ID}..."
        echo "Try: (${count}/${timeout})"
        sleep 10
        if [[ $count -eq timeout ]];then
            echo "Timeout waiting for cleaning the Route53 DNS records"
```

```
        exit 1
    fi
    count=$((count+1))
    ROUTES=$(aws route53 list-resource-record-sets --hosted-zone-id ${ZONE_ID} --max-items 10000 --output json | grep -c ${EXTERNAL_DNS_DOMAIN})
  done
}

# SAMPLE: clean_routes "<HC ControlPlane Namespace>" "<AWS_ZONE_ID>"
clean_routes "${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}" "${AWS_ZONE_ID}"
```

**検証**

すべての OpenShift Container Platform オブジェクトと S3 バケットをチェックし、すべてが想定どおりであることを確認します。

**次のステップ**

ホストされたクラスターを復元します。

### 4.2.4. ホストされたクラスターの復元

バックアップしたすべてのオブジェクトを収集し、宛先管理クラスターに復元します。

**前提条件**

ソース管理クラスターからデータをバックアップしている。

**ヒント**

宛先管理クラスターの **kubeconfig** ファイルが、**KUBECONFIG** 変数に設定されているとおりに、あるいは、スクリプトを使用する場合は **MGMT2_KUBECONFIG** 変数に設定されているとおりに配置されていることを確認します。**export KUBECONFIG=<Kubeconfig FilePath>** を使用するか、スクリプトを使用する場合は **export KUBECONFIG=${MGMT2_KUBECONFIG}** を使用します。

**手順**

1. 以下のコマンドを入力して、新しい管理クラスターに、復元するクラスターの namespace が含まれていないことを確認します。

```
# Just in case
export KUBECONFIG=${MGMT2_KUBECONFIG}
BACKUP_DIR=${HC_CLUSTER_DIR}/backup

# Namespace deletion in the destination Management cluster
$ oc delete ns ${HC_CLUSTER_NS} || true
$ oc delete ns ${HC_CLUSTER_NS}-{HC_CLUSTER_NAME} || true
```

2. 以下のコマンドを入力して、削除された namespace を再作成します。

```
# Namespace creation
$ oc new-project ${HC_CLUSTER_NS}
$ oc new-project ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}
```

3. 次のコマンドを入力して、HC namespace のシークレットを復元します。

```
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/secret-*
```

4. 以下のコマンドを入力して、**HostedCluster** コントロールプレーン namespace のオブジェクトを復元します。

```
# Secrets
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/secret-*

# Cluster
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/hcp-*
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/cl-*
```

5. ノードとノードプールを復元して AWS インスタンスを再利用する場合は、次のコマンドを入力して、HC コントロールプレーン namespace のオブジェクトを復元します。

```
# AWS
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/awscl-*
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/awsmt-*
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/awsm-*

# Machines
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machinedeployment-*
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machineset-*
$ oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machine-*
```

6. 次の bash スクリプトを実行して、etcd データとホストされたクラスターを復元します。

```
ETCD_PODS="etcd-0"
if [ "${CONTROL_PLANE_AVAILABILITY_POLICY}" = "HighlyAvailable" ]; then
  ETCD_PODS="etcd-0 etcd-1 etcd-2"
fi

HC_RESTORE_FILE=${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/hc-
${HC_CLUSTER_NAME}-restore.yaml
HC_BACKUP_FILE=${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/hc-
${HC_CLUSTER_NAME}.yaml
HC_NEW_FILE=${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/hc-
${HC_CLUSTER_NAME}-new.yaml
cat ${HC_BACKUP_FILE} > ${HC_NEW_FILE}
cat > ${HC_RESTORE_FILE} <<EOF
    restoreSnapshotURL:
EOF

for POD in ${ETCD_PODS}; do
  # Create a pre-signed URL for the etcd snapshot
  ETCD_SNAPSHOT="s3://${BUCKET_NAME}/${HC_CLUSTER_NAME}-${POD}-
```

```
snapshot.db"
  ETCD_SNAPSHOT_URL=$(AWS_DEFAULT_REGION=${MGMT2_REGION} aws s3
presign ${ETCD_SNAPSHOT})

  # FIXME no CLI support for restoreSnapshotURL yet
  cat >> ${HC_RESTORE_FILE} <<EOF
    - "${ETCD_SNAPSHOT_URL}"
EOF
done

cat ${HC_RESTORE_FILE}

if ! grep ${HC_CLUSTER_NAME}-snapshot.db ${HC_NEW_FILE}; then
  sed -i '' -e "/type: PersistentVolume/r ${HC_RESTORE_FILE}" ${HC_NEW_FILE}
  sed -i '' -e '/pausedUntil:/d' ${HC_NEW_FILE}
fi

HC=$(oc get hc -n ${HC_CLUSTER_NS} ${HC_CLUSTER_NAME} -o name || true)
if [[ ${HC} == "" ]];then
    echo "Deploying HC Cluster: ${HC_CLUSTER_NAME} in ${HC_CLUSTER_NS}
namespace"
    oc apply -f ${HC_NEW_FILE}
else
    echo "HC Cluster ${HC_CLUSTER_NAME} already exists, avoiding step"
fi
```

7. ノードとノードプールを復元して AWS インスタンスを再利用する場合は、次のコマンドを入力してノードプールを復元します。

```
oc apply -f ${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}/np-*
```

**検証**

- ノードが完全に復元されたことを確認するには、次の関数を使用します。

```
timeout=40
count=0
NODE_STATUS=$(oc get nodes --kubeconfig=${HC_KUBECONFIG} | grep -v NotReady |
grep -c "worker") || NODE_STATUS=0

while [ ${NODE_POOL_REPLICAS} != ${NODE_STATUS} ]
do
    echo "Waiting for Nodes to be Ready in the destination MGMT Cluster:
${MGMT2_CLUSTER_NAME}"
    echo "Try: (${count}/${timeout})"
    sleep 30
    if [[ $count -eq timeout ]];then
        echo "Timeout waiting for Nodes in the destination MGMT Cluster"
        exit 1
    fi
    count=$((count+1))
    NODE_STATUS=$(oc get nodes --kubeconfig=${HC_KUBECONFIG} | grep -v NotReady |
grep -c "worker") || NODE_STATUS=0
done
```

### 次のステップ

クラスターをシャットダウンして削除します。

## 4.2.5. ホストされたクラスターのソース管理クラスターからの削除

ホストされたクラスターをバックアップして宛先管理クラスターに復元した後、ソース管理クラスターのホストされたクラスターをシャットダウンして削除します。

### 前提条件

データをバックアップし、ソース管理クラスターに復元している。

### ヒント

宛先管理クラスターの **kubeconfig** ファイルが、**KUBECONFIG** 変数に設定されているとおりに、あるいは、スクリプトを使用する場合は **MGMT_KUBECONFIG** 変数に設定されているとおりに配置されていることを確認します。**export KUBECONFIG=<Kubeconfig FilePath>** を使用するか、スクリプトを使用する場合は **export KUBECONFIG=${MGMT_KUBECONFIG}** を使用します。

### 手順

1. 以下のコマンドを入力して、**deployment** および **statefulset** オブジェクトをスケーリングします。

```
# Just in case
export KUBECONFIG=${MGMT_KUBECONFIG}

# Scale down deployments
oc scale deployment -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --replicas=0 --all
oc scale statefulset.apps -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --replicas=0 --
all
sleep 15
```

2. 次のコマンドを入力して、**NodePool** オブジェクトを削除します。

```
NODEPOOLS=$(oc get nodepools -n ${HC_CLUSTER_NS} -o=jsonpath='{.items[?
(@.spec.clusterName=="'${HC_CLUSTER_NAME}'")].metadata.name}')
if [[ ! -z "${NODEPOOLS}" ]];then
    oc patch -n "${HC_CLUSTER_NS}" nodepool ${NODEPOOLS} --type=json --patch='[ {
"op":"remove", "path": "/metadata/finalizers" }]'
    oc delete np -n ${HC_CLUSTER_NS} ${NODEPOOLS}
fi
```

3. 次のコマンドを入力して、**machine** および **machineset** オブジェクトを削除します。

```
# Machines
for m in $(oc get machines -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -o name); do
    oc patch -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} ${m} --type=json --patch='[ {
"op":"remove", "path": "/metadata/finalizers" }]' || true
    oc delete -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} ${m} || true
done

oc delete machineset -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --all || true
```

4. 次のコマンドを入力して、クラスターオブジェクトを削除します。

```
# Cluster
C_NAME=$(oc get cluster -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -o name)
oc patch -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} ${C_NAME} --type=json --
patch='[ { "op":"remove", "path": "/metadata/finalizers" }]'
oc delete cluster.cluster.x-k8s.io -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} --all
```

5. 次のコマンドを入力して、AWS マシン (Kubernetes オブジェクト) を削除します。実際の AWS マシンの削除について心配する必要はありません。クラウドインスタンスへの影響はありません。

```
# AWS Machines
for m in $(oc get awsmachine.infrastructure.cluster.x-k8s.io -n ${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME} -o name)
do
    oc patch -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} ${m} --type=json --patch='[ {
"op":"remove", "path": "/metadata/finalizers" }]' || true
    oc delete -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} ${m} || true
done
```

6. 次のコマンドを入力して、**HostedControlPlane** および **ControlPlane** HC namespace オブジェクトを削除します。

```
# Delete HCP and ControlPlane HC NS
oc patch -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}
hostedcontrolplane.hypershift.openshift.io ${HC_CLUSTER_NAME} --type=json --patch='[ {
"op":"remove", "path": "/metadata/finalizers" }]'
oc delete hostedcontrolplane.hypershift.openshift.io -n ${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME} --all
oc delete ns ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} || true
```

7. 次のコマンドを入力して、**HostedCluster** および HC namespace オブジェクトを削除します。

```
# Delete HC and HC Namespace
oc -n ${HC_CLUSTER_NS} patch hostedclusters ${HC_CLUSTER_NAME} -p '{"metadata":
{"finalizers":null}}' --type merge || true
oc delete hc -n ${HC_CLUSTER_NS} ${HC_CLUSTER_NAME}  || true
oc delete ns ${HC_CLUSTER_NS} || true
```

**検証**

- すべてが機能することを確認するには、次のコマンドを入力します。

```
# Validations
export KUBECONFIG=${MGMT2_KUBECONFIG}

oc get hc -n ${HC_CLUSTER_NS}
oc get np -n ${HC_CLUSTER_NS}
oc get pod -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}
oc get machines -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME}

# Inside the HostedCluster
```

```
export KUBECONFIG=${HC_KUBECONFIG}
oc get clusterversion
oc get nodes
```

### 次のステップ

ホストされたクラスター内の OVN Pod を削除して、新しい管理クラスターで実行される新しい OVN コントロールプレーンに接続できるようにします。

1. ホストされたクラスターの kubeconfig パスを使用して **KUBECONFIG** 環境変数を読み込みます。
2. 以下のコマンドを入力します。

```
$ oc delete pod -n openshift-ovn-kubernetes --all
```

## 4.2.6. ホストされたクラスターをバックアップおよび復元するためのスクリプトの実行

ホストされたクラスターをバックアップし、AWS の同じリージョン内で復元するプロセスを迅速化するために、スクリプトを修正して実行できます。

### 手順

1. 次のスクリプトの変数を独自の情報に置き換えます。

```
# Fill the Common variables to fit your environment, this is just a sample
SSH_KEY_FILE=${HOME}/.ssh/id_rsa.pub
BASE_PATH=${HOME}/hypershift
BASE_DOMAIN="aws.sample.com"
PULL_SECRET_FILE="${HOME}/pull_secret.json"
AWS_CREDS="${HOME}/.aws/credentials"
CONTROL_PLANE_AVAILABILITY_POLICY=SingleReplica
HYPERSHIFT_PATH=${BASE_PATH}/src/hypershift
HYPERSHIFT_CLI=${HYPERSHIFT_PATH}/bin/hypershift
HYPERSHIFT_IMAGE=${HYPERSHIFT_IMAGE:-"quay.io/${USER}/hypershift:latest"}
NODE_POOL_REPLICAS=${NODE_POOL_REPLICAS:-2}

# MGMT Context
MGMT_REGION=us-west-1
MGMT_CLUSTER_NAME="${USER}-dev"
MGMT_CLUSTER_NS=${USER}
MGMT_CLUSTER_DIR="${BASE_PATH}/hosted_clusters/${MGMT_CLUSTER_NS}-
${MGMT_CLUSTER_NAME}"
MGMT_KUBECONFIG="${MGMT_CLUSTER_DIR}/kubeconfig"

# MGMT2 Context
MGMT2_CLUSTER_NAME="${USER}-dest"
MGMT2_CLUSTER_NS=${USER}
MGMT2_CLUSTER_DIR="${BASE_PATH}/hosted_clusters/${MGMT2_CLUSTER_NS}-
${MGMT2_CLUSTER_NAME}"
MGMT2_KUBECONFIG="${MGMT2_CLUSTER_DIR}/kubeconfig"

# Hosted Cluster Context
HC_CLUSTER_NS=clusters
HC_REGION=us-west-1
```

```
HC_CLUSTER_NAME="${USER}-hosted"
HC_CLUSTER_DIR="${BASE_PATH}/hosted_clusters/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}"
HC_KUBECONFIG="${HC_CLUSTER_DIR}/kubeconfig"
BACKUP_DIR=${HC_CLUSTER_DIR}/backup

BUCKET_NAME="${USER}-hosted-${MGMT_REGION}"

# DNS
AWS_ZONE_ID="Z026552815SS3YPH9H6MG"
EXTERNAL_DNS_DOMAIN="guest.jpdv.aws.kerbeross.com"
```

2. スクリプトをローカルファイルシステムに保存します。

3. 次のコマンドを入力して、スクリプトを実行します。

```
source <env_file>
```

ここの **env_file** は、スクリプトを保存したファイルの名前になります。

移行スクリプトは、https://github.com/openshift/hypershift/blob/main/contrib/migration/migrate-hcp.sh のリポジトリーで管理されています。