OpenShift Container Platform 4.16

Hosted Control Plane

OpenShift Container Platform で Hosted Control Plane を使用する

Last Updated: 2025-12-08

OpenShift Container Platform 4.16 Hosted Control Plane

OpenShift Container Platform で Hosted Control Plane を使用する

Legal Notice

Abstract

このドキュメントでは、OpenShift Container Platform の Hosted Control Plane を管理するための手順を説明します。Hosted Control Plane 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用意することなく、ホスティングクラスター上の Pod としてコントロールプレーンを作成できます。

Table of Contents

第1章 HOSTED CONTROL PLANE の概要

OpenShift Container Platform クラスターは、スタンドアロンまたは Hosted Control Plane 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。スタンドアロン構成では、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 Hosted Control Plane を使用すると、各コントロールプレーンに専用の仮想マシンまたは物理マシンを用意する必要なく、ホスティングクラスター上の Pod としてコントロールプレーンを作成できます。

1.1. HOSTED CONTROL PLANE の一般的な概念とペルソナの用語集

OpenShift Container Platform の Hosted Control Plane を使用する場合は、その主要な概念と関連するペルソナを理解することが重要です。

1.1.1. 概念

ホステッドクラスター

コントロールプレーンと API エンドポイントが管理クラスターでホストされている OpenShift Container Platform クラスター。ホステッドクラスターには、コントロールプレーンとそれに対応するデータプレーンが含まれます。

ホステッドクラスターのインフラストラクチャー

テナントまたはエンドユーザーのクラウドアカウントに存在するネットワーク、コンピュート、およびストレージリソース。

Hosted Control Plane

管理クラスターで実行される OpenShift Container Platform コントロールプレーン。ホステッドクラスターの API エンドポイントによって公開されます。コントロールプレーンのコンポーネントには、etcd、Kubernetes API サーバー、Kubernetes コントローラーマネージャー、および VPN が含まれます。

ホスティングクラスター

管理クラスター を参照してください。

マネージドクラスター

ハブクラスターが管理するクラスター。この用語は、Red Hat Advanced Cluster Management で multicluster engine for Kubernetes Operator が管理するクラスターライフサイクル特有の用語です。マネージドクラスターは、**管理クラスター** とは異なります。詳細は、マネージドクラスター を参照してください。

管理クラスター

HyperShift Operator がデプロイされる OpenShift Container Platform クラスター。ホステッドクラスターのコントロールプレーンをホストします。管理クラスターは **ホスティングクラスター** と同義です。

管理クラスターのインフラストラクチャー

管理クラスターのネットワーク、コンピュート、およびストレージリソース。

ノードプール

コンピュートノードを含むリソース。コントロールプレーンにはノードプールが含まれます。コンピュートノードはアプリケーションとワークロードを実行します。

1.1.2. ペルソナ

クラスターインスタンス管理者

このロールを引き受けるユーザーは、スタンドアロン OpenShift Container Platform の管理者と同

等です。このユーザーには、プロビジョニングされたクラスター内で **cluster-admin** ロールがありますが、クラスターがいつ、どのように更新または設定されるかを制御できない可能性があります。このユーザーは、クラスターに投影された設定を表示するための読み取り専用アクセス権を持っている可能性があります。

クラスターインスタンスユーザー

このロールを引き受けるユーザーは、スタンドアロン OpenShift Container Platform の開発者と同等です。このユーザーには、OperatorHub またはマシンに対するビューがありません。

クラスターサービスコンシューマー

このロールを引き受けるユーザーは、コントロールプレーンとワーカーノードを要求したり、更新を実行したり、外部化された設定を変更したりできます。通常、このユーザーはクラウド認証情報やインフラストラクチャー暗号化キーを管理したりアクセスしたりしません。クラスターサービスのコンシューマーペルソナは、ホステッドクラスターを要求し、ノードプールと対話できます。このロールを引き受けるユーザーには、論理境界内でホステッドクラスターとノードプールを作成、読み取り、更新、または削除するための RBAC があります。

クラスターサービスプロバイダー

このロールを引き受けるユーザーは通常、管理クラスター上で **cluster-admin** ロールを持ち、HyperShift Operator とテナントのホステッドクラスターのコントロールプレーンの可用性を監視および所有するための RBAC を持っています。クラスターサービスプロバイダーのペルソナは、次の例を含むいくつかのアクティビティーを担当します。

- コントロールプレーンの可用性、稼働時間、安定性を確保するためのサービスレベルオブジェクトの所有
- コントロールプレーンをホストするための管理クラスターのクラウドアカウントの設定
- 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の設定 (利用可能なコンピュートリソースのホスト認識を含む)

1.2. HOSTED CONTROL PLANE の概要

Red Hat OpenShift Container Platform の Hosted Control Plane を使用すると、管理コストを削減し、クラスターのデプロイ時間を最適化し、管理とワークロードの問題を分離して、アプリケーションに集中できるようになります。

Hosted Control Plane は、次のプラットフォームで Kubernetes Operator バージョン 2.0 以降のマルチクラスターエンジン を使用することで利用できます。

- Agent プロバイダーを使用したベアメタル
- OpenShift Virtualization。接続環境では一般提供機能として、非接続環境ではテクノロジープレビュー機能として提供されます。
- Amazon Web Services (AWS) (テクノロジープレビュー機能)
- IBM Z (テクノロジープレビュー機能)
- IBM Power (テクノロジープレビュー機能)

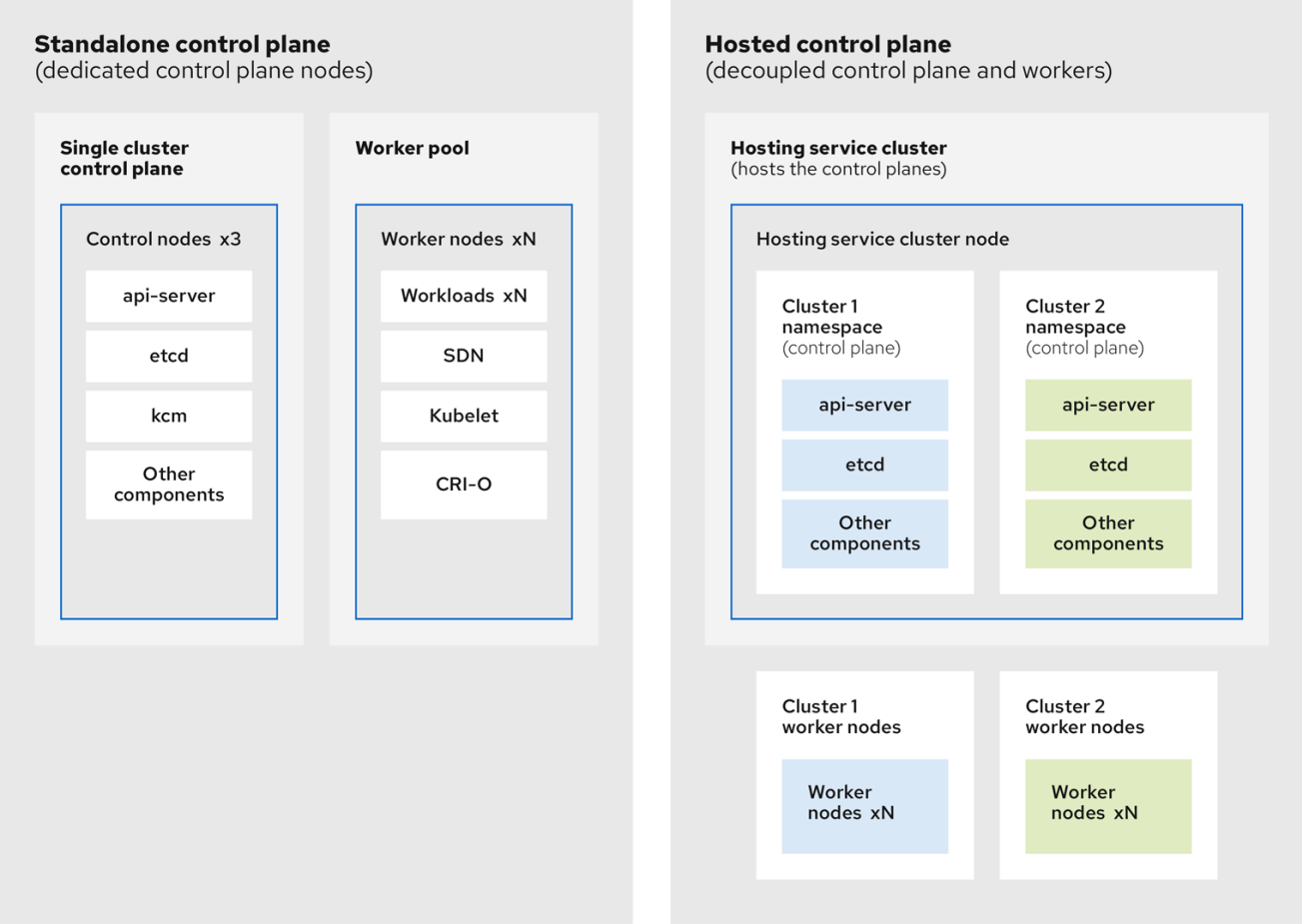
1.2.1. Hosted Control Plane のアーキテクチャー

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構成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ト、ストレージ、および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。



272_OpenShift_1122

1.2.2. Hosted Control Plane の利点

OpenShift Container Platform の Hosted Control Plane 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、その他のさまざまなメリットも享受できます。

- 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。

- Hosted Control Plane を使用すると、より少ないノードで多数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 で構成されるため、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ーンに移動できるため、解決までの時間 (TTR) が短縮され、生産性が向上します。

1.3. HOSTED CONTROL PLANE と OPENSHIFT CONTAINER PLATFORM の違い

Hosted Control Plane は、OpenShift Container Platform の 1 つのフォームファクターです。ホステッドクラスターとスタンドアロンの OpenShift Container Platform クラスターは、設定と管理が異なります。OpenShift Container Platform と Hosted Control Plane の違いを理解するには、次の表を参照してください。

1.3.1. クラスターの作成とライフサイクル

OpenShift Container Platform	Hosted Control Plane
openshift-install バイナリーまたは Assisted Installer を使用して、スタンドアロンの OpenShift Container Platform クラスターをインストールします。	既存の OpenShift Container Platform クラスターに、**HostedCluster** や **NodePool** などの **hypershift.openshift.io** API リソースを使用して、ホステッドクラスターをインストールします。

1.3.2. Cluster configuration

OpenShift Container Platform	Hosted Control Plane
config.openshift.io API グループを使用して、認証、API サーバー、プロキシーなど、クラスタースコープのリソースを設定します。	コントロールプレーンに影響するリソースを、**HostedCluster** リソースで設定します。

1.3.3. etcd 暗号化

OpenShift Container Platform	Hosted Control Plane
APIServer リソースと AES-GCM または AES-CBC を使用して、etcd 暗号化を設定します。詳細は、「etcd 暗号化の有効化」を参照してください。	**SecretEncryption** フィールドの **HostedCluster** リソースと Amazon Web Services の AES-CBC または KMS を使用して、etcd 暗号化を設定します。

1.3.4. Operator とコントロールプレーン

OpenShift Container Platform	Hosted Control Plane
スタンドアロンの OpenShift Container Platform クラスターには、コントロールプレーンコンポーネントごとに個別の Operator が含まれています。	ホステッドクラスターには、管理クラスターの Hosted Control Plane の namespace で実行される Control Plane Operator という名前の単一の Operator が含まれています。
etcd は、コントロールプレーンノードにマウントされたストレージを使用します。etcd のクラスター Operator が etcd を管理します。	etcd は、ストレージに永続ボリューム要求を使用し、Control Plane Operator によって管理されます。
Ingress Operator、ネットワーク関連の Operator、および Operator Lifecycle Manager (OLM) は、クラスター上で実行されます。	Ingress Operator、ネットワーク関連の Operator、および Operator Lifecycle Manager (OLM) は、管理クラスターの Hosted Control Plane の namespace で実行されます。
OAuth サーバーは、クラスター内で実行され、クラスター内のルートを通じて公開されます。	OAuth サーバーは、コントロールプレーン内で実行され、管理クラスター上のルート、ノードポート、またはロードバランサーを通じて公開されます。

1.3.5. 更新

OpenShift Container Platform	Hosted Control Plane
Cluster Version Operator (CVO) が更新プロセスをオーケストレーションし、**ClusterVersion** リソースを監視します。管理者と OpenShift コンポーネントは、**ClusterVersion** リソースを通じて CVO とやり取りできます。**oc adm upgrade** コマンドを実行すると、**ClusterVersion** リソースの **ClusterVersion.Spec.DesiredUpdate** フィールドが変更されます。	Hosted Control Plane を更新すると、**HostedCluster** および **NodePool** リソースの **.spec.release.image** フィールドが変更されます。**ClusterVersion** リソースへの変更はすべて無視されます。
OpenShift Container Platform クラスターを更新すると、コントロールプレーンとコンピュートマシンの両方が更新されます。	ホステッドクラスターを更新すると、コントロールプレーンのみが更新されます。ノードプールの更新は個別に実行します。

1.3.6. マシンの設定と管理

OpenShift Container Platform	Hosted Control Plane
MachineSets リソースが、**openshift-machine-api** namespace 内のマシンを管理します。	**NodePool** リソースが、管理クラスター上のマシンを管理します。
コントロールプレーンマシンのセットが利用可能です。	コントロールプレーンマシンのセットが存在しません。
MachineHealthCheck リソースを使用して、マシンのヘルスチェックを有効にします。	**NodePool** リソースの **.spec.management.autoRepair** フィールドを通じてマシンのヘルスチェックを有効にします。
ClusterAutoscaler および **MachineAutoscaler** リソースを使用して自動スケーリングを有効にします。	**NodePool** リソースの **spec.autoScaling** フィールドを通じて自動スケーリングを有効にします。
マシンとマシンセットがクラスター内で公開されます。	アップストリームの Cluster CAPI Operator からのマシン、マシンセット、およびマシンデプロイメントが、マシンを管理するために使用されます。ただし、これらはユーザーには公開されません。
クラスターを更新すると、すべてのマシンセットが自動的にアップグレードされます。	ホステッドクラスターの更新とは別にノードプールを更新します。
インプレースアップグレードだけがクラスターでサポートされています。	置換アップグレードとインプレースアップグレードの両方が、ホステッドクラスターでサポートされています。
Machine Config Operator がマシンの設定を管理します。	Hosted Control Plane には Machine Config Operator が存在しません。
MachineConfigPool セレクターから選択された **MachineConfig**、**KubeletConfig**、および **ContainerRuntimeConfig** リソースを使用して、マシンの Ignition を設定します。	**MachineConfig**、**KubeletConfig**、および **ContainerRuntimeConfig** リソースを、**NodePool** リソースの **spec.config** フィールドで参照される config map を通じて設定します。
Machine Config Daemon (MCD) が各ノードの設定の変更と更新を管理します。	インプレースアップグレードの場合、ノードプールコントローラーが、一度だけ実行され、設定に基づいてマシンを更新する Pod を作成します。
SR-IOV Operator などのマシン設定リソースを変更できます。	マシン設定リソースを変更することはできません。

1.3.7. ネットワーク

OpenShift Container Platform	Hosted Control Plane
Kube API サーバーとノードが同じ Virtual Private Cloud (VPC) 内に存在するため、Kube API サーバーはノードと直接通信します。	Kube API サーバーは Konnectivity を介してノードと通信します。Kube API サーバーとノードは、別々の Virtual Private Cloud (VPC) 内に存在します。
ノードは内部ロードバランサーを介して Kube API サーバーと通信します。	ノードは外部ロードバランサーまたはノードポートを介して Kube API サーバーと通信します。

1.3.8. Web コンソール

OpenShift Container Platform	Hosted Control Plane
Web コンソールにコントロールプレーンのステータスが表示されます。	Web コンソールにコントロールプレーンのステータスが表示されません。
Web コンソールを使用してクラスターを更新できます。	Web コンソールを使用してホステッドクラスターを更新することはできません。
Web コンソールにマシンなどのインフラストラクチャーリソースが表示されます。	Web コンソールにインフラストラクチャーリソースが表示されません。
Web コンソールを使用して、**MachineConfig** リソースを通じてマシンを設定できます。	Web コンソールを使用してマシンを設定することはできません。

関連情報

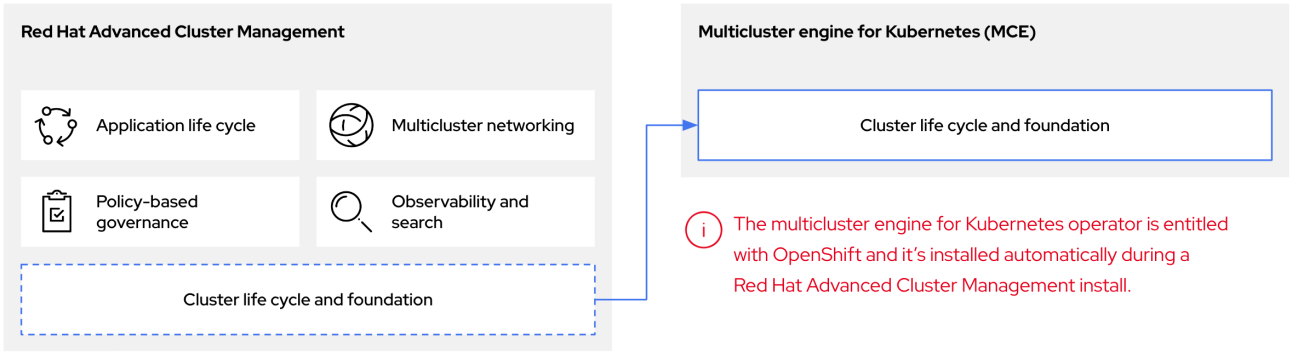
- etcd 暗号化の有効化

1.4. HOSTED CONTROL PLANE、MULTICLUSTER ENGINE OPERATOR、および RHACM の関係

Hosted Control Plane は、multicluster engine for Kubernetes Operator を使用して設定できます。マルチクラスターエンジンは Red Hat Advanced Cluster Management (RHACM) の不可欠な要素であり、RHACM ではデフォルトで有効になっています。multicluster engine Operator のクラスターライフサイクルにより、さまざまなインフラストラクチャークラウドプロバイダー、プライベートクラウド、オンプレミスデータセンターにおける Kubernetes クラスターの作成、インポート、管理、破棄のプロセスが定義されます。

multicluster engine Operator は、OpenShift Container Platform および RHACM ハブクラスターにクラスター管理機能を提供するクラスターライフサイクル Operator です。multicluster engine Operator は、クラスター群の管理を強化し、クラウドとデータセンター全体の OpenShift Container Platform クラスターのライフサイクル管理を支援します。

図1.1 クラスターライフサイクルと基盤



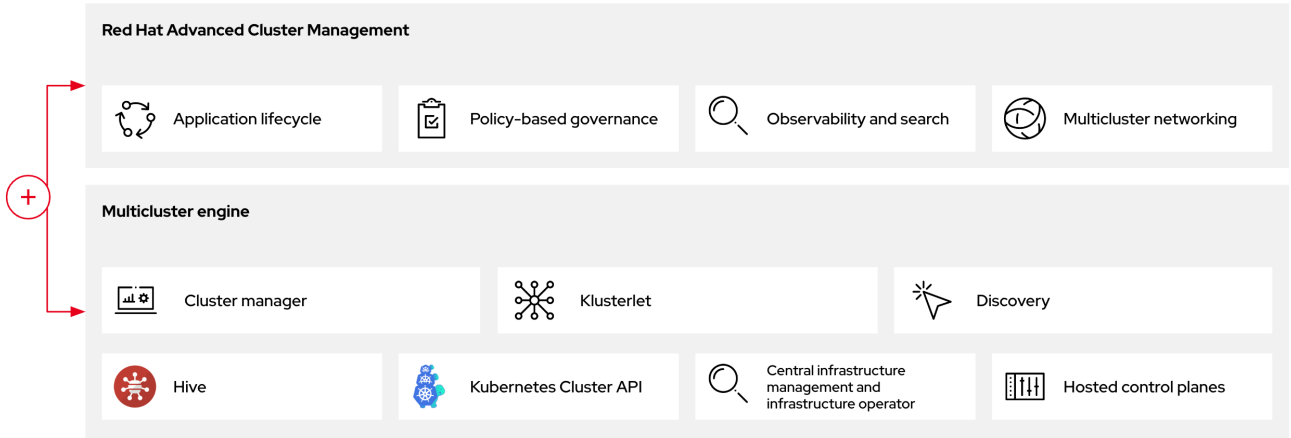
OpenShift Container Platform の multicluster engine Operator は、スタンドアロンクラスターマネージャーとして、または RHACM ハブクラスターの一部として使用できます。

ヒント

管理クラスターはホスティングクラスターとも呼ばれます。

OpenShift Container Platform クラスターは、スタンドアロンまたは Hosted Control Plane 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 Hosted Control Plane を使用すると、各コントロールプレーンに専用の仮想マシンまたは物理マシンを用意する必要なく、管理クラスター上の Pod としてコントロールプレーンを作成できます。

図1.2 RHACM と multicluster engine Operator の概要図



1.5. HOSTED CONTROL PLANE 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、Hosted Control Plane の 2 つのコンポーネントがリリースされます。

- HyperShift Operator
- **hcp** 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 は、**hypershift** namespace に **supported-versions** config map

を作成します。この config map には、サポートされているホステッドクラスターのバージョンが含まれています。

同じ管理クラスター上で異なるバージョンのコントロールプレーンをホストできます。

supported-versions config map オブジェクトの例

```
apiVersion: v1
data:
  supported-versions: '{"versions":["4.16"]}'
kind: ConfigMap
metadata:
  labels:
    hypershift.openshift.io/supported-versions: "true"
  name: supported-versions
  namespace: hypershift
```

hcp CLI を使用してホストされたクラスターを作成できます。

HostedCluster や **NodePool** などの **hypershift.openshift.io** API リソースを使用して、大規模な OpenShift Container Platform クラスターを作成および管理できます。**HostedCluster** リソースには、コントロールプレーンと共通データプレーンの設定が含まれます。**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。

関連情報

- ホステッドクラスターにおけるノードのチューニング設定
- カーネルブートパラメーターを設定することによる、ホステッドクラスターの高度なノードチューニング

第2章 HOSTED CONTROL PLANE の使用開始

OpenShift Container Platform の Hosted Control Plane 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。

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. ベアメタル

- Hosted Control Plane のサイジングに関するガイダンス
- Hosted Control Plane コマンドラインインターフェイスのインストール
- ホステッドクラスターのワークロードの分散
- ベアメタルのファイアウォールとポートの要件
- ベアメタルインフラストラクチャーの要件: ベアメタル上にホストされたクラスターを作成するためのインフラストラクチャー要件を確認します。
- ベアメタルでの Hosted Control Plane クラスターの設定
  - DNS を設定する
  - ホストされたクラスターを作成し、クラスターの作成を確認する
  - ホストされたクラスターの **NodePool** オブジェクトをスケーリングする
  - ホストされたクラスターの ingress トラフィックを処理する
  - ホストされたクラスターのノードの自動スケーリングを有効にする
- 非接続環境での Hosted Control Plane の設定
- ベアメタル上のホストされたクラスターを破棄するには、ベアメタル上のホステッドクラスターの破棄 の手順に従ってください。
- Hosted Control Plane 機能を無効にする場合は、Hosted Control Plane 機能の無効化 を参照してください。

2.2. OPENSHIFT VIRTUALIZATION

- Hosted Control Plane のサイジングに関するガイダンス
- Hosted Control Plane コマンドラインインターフェイスのインストール
- ホステッドクラスターのワークロードの分散
- OpenShift Virtualization 上での Hosted Control Plane クラスターの管理 : 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。
- 非接続環境での Hosted Control Plane の設定
- OpenShift Virtualization 上でホストされているクラスターを破棄するには、OpenShift Virtualization 上のホステッドクラスターの破棄 の手順に従ってください。

- Hosted Control Plane 機能を無効にする場合は、Hosted Control Plane 機能の無効化 を参照してください。

2.3. AMAZON WEB SERVICES (AWS)

- AWS インフラストラクチャー要件: AWS 上にホストされたクラスターを作成するためのインフラストラクチャー要件を確認します。
- AWS でホストされたコントロールプレーンクラスターを設定する: AWS でホストされたコントロールプレーン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、ホストされたクラスターのデプロイが含まれます。
- Hosted Control Plane 用 SR-IOV Operator のデプロイ : ホスティングサービスクラスターを設定してデプロイした後、ホストされたクラスター上で Single Root I/O Virtualization (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。
- AWS でホストされているクラスターを破棄するには、AWS 上のホステッドクラスターの破棄 の手順に従ってください。
- Hosted Control Plane 機能を無効にする場合は、Hosted Control Plane 機能の無効化 を参照してください。

2.4. IBM Z

重要

IBM Z プラットフォーム上の Hosted Control Plane 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 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

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

- テクノロジープレビュー機能のサポート範囲

- Hosted Control Plane コマンドラインインターフェイスのインストール
- IBM Z コンピュートノード用の x86 ベアメタル上でホスティングクラスターを設定する (テクノロジープレビュー)

2.5. IBM POWER

重要

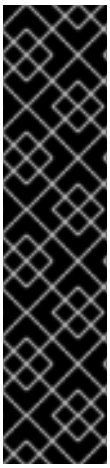
IBM Power プラットフォーム上の Hosted Control Plane は、テクノロジープレビュー機能としてのみ利用できます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

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

- テクノロジープレビュー機能のサポート範囲

- Hosted Control Plane コマンドラインインターフェイスのインストール
- 64 ビット x86 OpenShift Container Platform クラスターでのホスティングクラスターの設定による、IBM Power コンピュートノードの Hosted Control Plane の作成 (テクノロジープレビュー)

2.6. 非ベアメタルエージェントマシン



重要

非ベアメタルエージェントマシンを使用する Hosted Control Plane クラスターは、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

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

- テクノロジープレビュー機能のサポート範囲

- Hosted Control Plane コマンドラインインターフェイスのインストール
- 非ベアメタルエージェントマシンを使用した Hosted Control Plane クラスターの設定 (テクノロジープレビュー)
- ベアメタルエージェント以外のマシンでホストされているクラスターを破棄するには、非ベアメタルエージェントマシン上のホステッドクラスターの破棄 の手順に従ってください。
- Hosted Control Plane 機能を無効にする場合は、Hosted Control Plane 機能の無効化 を参照してください。

第3章 HOSTED CONTROL PLANE の認証と認可

OpenShift Container Platform のコントロールプレーンには、組み込みの OAuth サーバーが含まれています。OAuth アクセストークンを取得することで、OpenShift Container Platform API に対して認証できます。ホステッドクラスターを作成した後に、アイデンティティープロバイダーを指定して OAuth を設定できます。

3.1. CLI を使用してホストされたクラスターの OAUTH サーバーを設定する

OpenID Connect アイデンティティープロバイダー (**oidc**) を使用して、ホストされたクラスターの内部 OAuth サーバーを設定できます。

サポートされている次のアイデンティティープロバイダーに対して OAuth を設定できます。

- **oidc**
- **htpasswd**
- **keystone**
- **ldap**
- **basic-authentication**
- **request-header**
- **github**
- **gitlab**
- **google**

OAuth 設定にアイデンティティープロバイダーを追加すると、デフォルトの **kubeadmin** ユーザープロバイダーが削除されます。

注記

アイデンティティープロバイダーを設定するときは、ホステッドクラスターに少なくとも 1 つの **NodePool** レプリカを事前に設定する必要があります。DNS 解決のトラフィックはワーカーノードを介して送信されます。**htpasswd** および **request-header** アイデンティティープロバイダー用に、**NodePool** レプリカを事前に設定する必要はありません。

前提条件

- ホステッドクラスターを作成した。

手順

1. 次のコマンドを実行して、ホスティングクラスターで **HostedCluster** カスタムリソース (CR) を編集します。

```
$ oc edit hostedcluster <hosted_cluster_name> -n <hosted_cluster_namespace>
```

2. 次の例を使用して、**HostedCluster** CR に OAuth 設定を追加します。

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: HostedCluster
metadata:
  name: <hosted_cluster_name> 1
  namespace: <hosted_cluster_namespace> 2
spec:
  configuration:
    oauth:
      identityProviders:
      - openID: 3
          claims:
            email: 4
              - <email_address>
            name: 5
              - <display_name>
            preferredUsername: 6
              - <preferred_username>
          clientID: <client_id> 7
          clientSecret:
            name: <client_id_secret_name> 8
          issuer: https://example.com/identity 9
        mappingMethod: lookup 10
        name: IAM
        type: OpenID
```

1. ホステッドクラスターの名前を指定します。
2. ホステッドクラスターの namespace を指定します。
3. このプロバイダー名はアイデンティティー要求の値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。プロバイダー名はリダイレクト URL の構築にも使用されます。
4. メールアドレスとして使用する属性のリストを定義します。
5. 表示名として使用する属性のリストを定義します。
6. 優先ユーザー名として使用する属性のリストを定義します。
7. OpenID プロバイダーに登録されたクライアントの ID を定義します。このクライアントを URL **https://oauth-openshift.apps.<cluster_name>.<cluster_domain>/oauth2callback/<idp_provider_name>** にリダイレクトできるようにする必要があります。
8. OpenID プロバイダーに登録されたクライアントのシークレットを定義します。
9. OpenID の仕様で説明されている Issuer Identifier。クエリーまたはフラグメントコンポーネントのない **https** を使用する必要があります。
10. このプロバイダーのアイデンティティーと **User** オブジェクトの間でマッピングを確立する方法を制御するマッピング方法を定義します。

3. 変更を適用するためにファイルを保存します。

3.2. WEB コンソールを使用してホステッドクラスターの OAUTH サーバーを設定する

OpenShift Container Platform Web コンソールを使用して、ホステッドクラスターの内部 OAuth サーバーを設定できます。

サポートされている次のアイデンティティープロバイダーに対して OAuth を設定できます。

- **oidc**
- **htpasswd**
- **keystone**
- **ldap**
- **basic-authentication**
- **request-header**
- **github**
- **gitlab**
- **google**

OAuth 設定にアイデンティティープロバイダーを追加すると、デフォルトの **kubeadmin** ユーザープロバイダーが削除されます。

注記

アイデンティティープロバイダーを設定するときは、ホステッドクラスターに少なくとも 1 つの **NodePool** レプリカを事前に設定する必要があります。DNS 解決のトラフィックはワーカーノードを介して送信されます。**htpasswd** および **request-header** アイデンティティープロバイダー用に、**NodePool** レプリカを事前に設定する必要はありません。

前提条件

- **cluster-admin** 権限を持つユーザーとしてログインしている。
- ホステッドクラスターを作成した。

手順

1. Home → API Explorer に移動します。
2. Filter by kind ボックスを使用して、**HostedCluster** リソースを検索します。
3. 編集する **HostedCluster** リソースをクリックします。
4. Instances タブをクリックします。

5. ホステッドクラスター名エントリーの横にあるオプションメニュー  をクリックし、**Edit HostedCluster** をクリックします。

6. YAML ファイルに OAuth 設定を追加します。

```
spec:
  configuration:
    oauth:
      identityProviders:
      - openID: (1)
          claims:
            email: (2)
              - <email_address>
            name: (3)
              - <display_name>
            preferredUsername: (4)
              - <preferred_username>
          clientID: <client_id> (5)
          clientSecret:
            name: <client_id_secret_name> (6)
          issuer: https://example.com/identity (7)
        mappingMethod: lookup (8)
        name: IAM
        type: OpenID
```

(1) このプロバイダー名はアイデンティティー要求の値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。プロバイダー名はリダイレクト URL の構築にも使用されます。

(2) メールアドレスとして使用する属性のリストを定義します。

(3) 表示名として使用する属性のリストを定義します。

(4) 優先ユーザー名として使用する属性のリストを定義します。

(5) OpenID プロバイダーに登録されたクライアントの ID を定義します。このクライアントを URL **https://oauth-openshift.apps.<cluster_name>.<cluster_domain>/oauth2callback/<idp_provider_name>** にリダイレクトできるようにする必要があります。

(6) OpenID プロバイダーに登録されたクライアントのシークレットを定義します。

(7) OpenID の仕様で説明されている Issuer Identifier。クエリーまたはフラグメントコンポーネントのない **https** を使用する必要があります。

(8) このプロバイダーのアイデンティティーと **User** オブジェクトの間でマッピングを確立する方法を制御するマッピング方法を定義します。

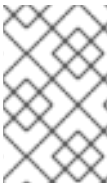
7. **Save** をクリックします。

関連情報

- サポートされているアイデンティティープロバイダーの詳細は、**認証および認可** の アイデンティティープロバイダー設定について を参照してください。

3.3. AWS 上のホステッドクラスターで CCO を使用してコンポーネントに IAM ロールを割り当てる

Amazon Web Services (AWS) 上のホステッドクラスターで Cloud Credential Operator (CCO) を使用して、権限が限られた短期間のセキュリティー認証情報を提供する IAM ロールをコンポーネントに割り当てることができます。デフォルトでは、CCO は Hosted Control Plane で実行されます。



注記

CCO は、AWS 上のホステッドクラスターでのみ手動モードをサポートします。デフォルトでは、ホステッドクラスターは手動モードに設定されます。管理クラスターでは、手動以外のモードを使用する場合があります。

3.4. AWS 上のホステッドクラスターで CCO のインストールを検証する

Hosted Control Plane で Cloud Credential Operator (CCO) が正しく実行されていることを確認できます。

前提条件

- Amazon Web Services (AWS) でホステッドクラスターを設定した。

手順

1. 次のコマンドを実行して、ホステッドクラスターで CCO が手動モードで設定されていることを確認します。

```
$ oc get cloudcredentials <hosted_cluster_name> -n <hosted_cluster_namespace> -o=jsonpath={.spec.credentialsMode}
```

予想される出力

```
Manual
```

2. 次のコマンドを実行して、**serviceAccountIssuer** リソースの値が空でないことを確認します。

```
$ oc get authentication cluster --kubeconfig <hosted_cluster_name>.kubeconfig -o jsonpath --template '{.spec.serviceAccountIssuer }'
```

出力例

```
https://aos-hypershift-ci-oidc-29999.s3.us-east-2.amazonaws.com/hypershift-ci-29999
```

3.5. OPERATOR が AWS STS を使用した CCO ベースのワークフローをサポートできるようにする

Operator Lifecycle Manager (OLM) で実行するプロジェクトを設計する Operator 作成者は、Cloud Credential Operator (CCO) をサポートするようにプロジェクトをカスタマイズすることで、STS 対応

の OpenShift Container Platform クラスター上で Operator が AWS に対して認証できるようにすることができます。

このメソッドでは、Operator が **CredentialsRequest** オブジェクトの作成を担当します。つまり、Operator がこれらのオブジェクトを作成するには RBAC 権限が必要です。次に、Operator は、結果として得られる **Secret** オブジェクトを読み取ることができなければなりません。

注記

デフォルトでは、Operator デプロイメントに関連する Pod は、結果として得られる **Secret** オブジェクトでサービスアカウントトークンを参照できるように、**serviceAccountToken** ボリュームをマウントします。

前提条件

- OpenShift Container Platform 4.14 以降
- STS モードのクラスター
- OLM ベースの Operator プロジェクト

手順

1. Operator プロジェクトの **ClusterServiceVersion** (CSV) オブジェクトを更新します。
   a. Operator が **CredentialsRequests** オブジェクトを作成する RBAC 権限を持っていることを確認します。

例3.1 clusterPermissions リストの例

```
# ...
install:
  spec:
    clusterPermissions:
    - rules:
      - apiGroups:
        - "cloudcredential.openshift.io"
        resources:
        - credentialsrequests
        verbs:
        - create
        - delete
        - get
        - list
        - patch
        - update
        - watch
```

   b. AWS STS を使用したこの CCO ベースのワークフロー方式のサポートを要求するために、次のアノテーションを追加します。

```
# ...
metadata:
 annotations:
```

```
    features.operators.openshift.io/token-auth-aws: "true"
```

2. Operator プロジェクトコードを更新します。

   a. **Subscription** オブジェクトによって Pod に設定された環境変数からロール ARN を取得します。以下に例を示します。

```
// Get ENV var
roleARN := os.Getenv("ROLEARN")
setupLog.Info("getting role ARN", "role ARN = ", roleARN)
webIdentityTokenPath := "/var/run/secrets/openshift/serviceaccount/token"
```

   b. パッチを適用して適用できる **CredentialsRequest** オブジェクトがあることを確認してください。以下に例を示します。

例3.2 CredentialsRequest オブジェクトの作成例

```
import (
  minterv1 "github.com/openshift/cloud-credential-
operator/pkg/apis/cloudcredential/v1"
  corev1 "k8s.io/api/core/v1"
  metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

var in = minterv1.AWSProviderSpec{
  StatementEntries: []minterv1.StatementEntry{
    {
      Action: []string{
        "s3:*",
      },
      Effect:   "Allow",
      Resource: "arn:aws:s3:*:*:*",
    },
  },
 STSIAMRoleARN: "<role_arn>",
}

var codec = minterv1.Codec
var ProviderSpec, _ = codec.EncodeProviderSpec(in.DeepCopyObject())

const (
  name      = "<credential_request_name>"
  namespace = "<namespace_name>"
)

var CredentialsRequestTemplate = &minterv1.CredentialsRequest{
  ObjectMeta: metav1.ObjectMeta{
     Name:      name,
     Namespace: "openshift-cloud-credential-operator",
  },
  Spec: minterv1.CredentialsRequestSpec{
    ProviderSpec: ProviderSpec,
    SecretRef: corev1.ObjectReference{
      Name:      "<secret_name>",
      Namespace: namespace,
    },
```

```
      ServiceAccountNames: []string{
        "<service_account_name>",
      },
      CloudTokenPath:   "",
    },
}
```

あるいは、YAML 形式の **CredentialsRequest** オブジェクトから開始している場合 (たとえば、Operator プロジェクトコードの一部として)、別の方法で処理することもできます。

例3.3 YAML フォームでの CredentialsRequest オブジェクト作成の例

```
// CredentialsRequest is a struct that represents a request for credentials
type CredentialsRequest struct {
  APIVersion string `yaml:"apiVersion"`
  Kind       string `yaml:"kind"`
  Metadata   struct {
    Name      string `yaml:"name"`
    Namespace string `yaml:"namespace"`
  } `yaml:"metadata"`
  Spec struct {
    SecretRef struct {
      Name      string `yaml:"name"`
      Namespace string `yaml:"namespace"`
    } `yaml:"secretRef"`
    ProviderSpec struct {
      APIVersion     string `yaml:"apiVersion"`
      Kind           string `yaml:"kind"`
      StatementEntries []struct {
        Effect   string   `yaml:"effect"`
        Action   []string `yaml:"action"`
        Resource string   `yaml:"resource"`
      } `yaml:"statementEntries"`
      STSIAMRoleARN   string `yaml:"stsIAMRoleARN"`
    } `yaml:"providerSpec"`

    // added new field
     CloudTokenPath   string `yaml:"cloudTokenPath"`
  } `yaml:"spec"`
}

// ConsumeCredsRequestAddingTokenInfo is a function that takes a YAML filename
and two strings as arguments
// It unmarshals the YAML file to a CredentialsRequest object and adds the token
information.
func ConsumeCredsRequestAddingTokenInfo(fileName, tokenString, tokenPath
string) (*CredentialsRequest, error) {
  // open a file containing YAML form of a CredentialsRequest
  file, err := os.Open(fileName)
  if err != nil {
    return nil, err
  }
  defer file.Close()
```

```
  // create a new CredentialsRequest object
  cr := &CredentialsRequest{}

  // decode the yaml file to the object
  decoder := yaml.NewDecoder(file)
  err = decoder.Decode(cr)
  if err != nil {
    return nil, err
  }

  // assign the string to the existing field in the object
  cr.Spec.CloudTokenPath = tokenPath

  // return the modified object
  return cr, nil
}
```

注記

CredentialsRequest オブジェクトを Operator バンドルに追加することは現在サポートされていません。

c. ロール ARN と Web ID トークンパスを認証情報リクエストに追加し、Operator の初期化中に適用します。

例3.4 Operator の初期化中に CredentialsRequest オブジェクトを適用する例

```
// apply credentialsRequest on install
credReq := credreq.CredentialsRequestTemplate
credReq.Spec.CloudTokenPath = webIdentityTokenPath

c := mgr.GetClient()
if err := c.Create(context.TODO(), credReq); err != nil {
  if !errors.IsAlreadyExists(err) {
    setupLog.Error(err, "unable to create CredRequest")
    os.Exit(1)
  }
}
```

d. 次の例に示すように、CCO から **Secret** オブジェクトが表示されるのを Operator が待機できるようにします。この処理は、Operator がリコンサイルする他の項目とともに呼び出されます。

例3.5 Secret オブジェクトを待機する例

```
// WaitForSecret is a function that takes a Kubernetes client, a namespace, and a v1
"k8s.io/api/core/v1" name as arguments
// It waits until the secret object with the given name exists in the given namespace
// It returns the secret object or an error if the timeout is exceeded
func WaitForSecret(client kubernetes.Interface, namespace, name string)
(*v1.Secret, error) {
  // set a timeout of 10 minutes
  timeout := time.After(10 * time.Minute) (1)
```

```
  // set a polling interval of 10 seconds
  ticker := time.NewTicker(10 * time.Second)

  // loop until the timeout or the secret is found
  for {
    select {
    case <-timeout:
      // timeout is exceeded, return an error
      return nil, fmt.Errorf("timed out waiting for secret %s in namespace %s", name,
namespace)
        // add to this error with a pointer to instructions for following a manual path to a
Secret that will work on STS
    case <-ticker.C:
      // polling interval is reached, try to get the secret
      secret, err := client.CoreV1().Secrets(namespace).Get(context.Background(),
name, metav1.GetOptions{})
      if err != nil {
        if errors.IsNotFound(err) {
          // secret does not exist yet, continue waiting
          continue
        } else {
          // some other error occurred, return it
          return nil, err
        }
      } else {
        // secret is found, return it
        return secret, nil
      }
    }
  }
}
```

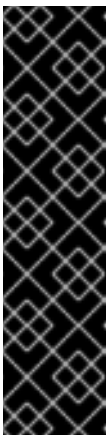
1 **timeout** 値は、CCO が追加された **CredentialsRequest** オブジェクトを検出して **Secret** オブジェクトを生成する速度の推定に基づいています。Operator がまだクラウドリソースにアクセスしていない理由を疑問に思っているクラスター管理者のために、時間を短縮するか、カスタムフィードバックを作成することを検討してください。

e. 認証情報リクエストから CCO によって作成されたシークレットを読み取り、そのシークレットのデータを含む AWS 設定ファイルを作成することで、AWS 設定をセットアップします。

例3.6 AWS 設定の作成例

```
func SharedCredentialsFileFromSecret(secret *corev1.Secret) (string, error) {
  var data []byte
  switch {
  case len(secret.Data["credentials"]) > 0:
     data = secret.Data["credentials"]
  default:
     return "", errors.New("invalid secret for aws credentials")
  }
```

```
    f, err := ioutil.TempFile("", "aws-shared-credentials")
    if err != nil {
        return "", errors.Wrap(err, "failed to create file for shared credentials")
    }
    defer f.Close()
    if _, err := f.Write(data); err != nil {
        return "", errors.Wrapf(err, "failed to write credentials to %s", f.Name())
    }
    return f.Name(), nil
}
```



重要

シークレットは存在すると想定されますが、このシークレットを使用する場合、CCO がシークレットを作成する時間を与えるために、Operator コードは待機して再試行する必要があります。

さらに、待機期間は最終的にタイムアウトになり、OpenShift Container Platform クラスターのバージョン、つまり CCO が、STS 検出による **CredentialsRequest** オブジェクトのワークフローをサポートしていない以前のバージョンである可能性があることをユーザーに警告します。このような場合は、別の方法を使用してシークレットを追加する必要があることをユーザーに指示します。

f. AWS SDK セッションを設定します。以下に例を示します。

例3.7 AWS SDK セッション設定の例

```
sharedCredentialsFile, err := SharedCredentialsFileFromSecret(secret)
if err != nil {
    // handle error
}
options := session.Options{
    SharedConfigState: session.SharedConfigEnable,
    SharedConfigFiles: []string{sharedCredentialsFile},
}
```

関連情報

- Cloud Credential Operator の Cloud Credential Operator リファレンスページ

第4章 HOSTED CONTROL PLANE のマシン設定の処理

スタンドアロン OpenShift Container Platform クラスターでは、マシン設定プールがノードのセットを管理します。**MachineConfigPool** カスタムリソース (CR) を使用してマシン設定を処理できます。

ヒント

NodePool CR の **nodepool.spec.config** フィールドで、任意の **machineconfiguration.openshift.io** リソースを参照できます。

ホストされたコントロールプレーンでは、**MachineConfigPool** CR は存在しません。ノードプールには、一連のコンピュートノードがあります。ノードプールを使用してマシン設定を処理できます。

4.1. 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。

```
$ oc edit nodepool <nodepool_name> --namespace <hosted_cluster_namespace>
```

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
# ...
  name: nodepool-1
  namespace: clusters
# ...
spec:
  config:
  - name: <configmap_name> 1
# ...
```

1 **<configmap_name>** は、設定マップの名前に置き換えます。

4.2. ノードプール内の KUBELET 設定を参照する

ノードプール内の kubelet 設定を参照するには、kubelet 設定を config map に追加してから、その config map を **NodePool** リソースに適用します。

手順

1. 次の情報を入力して、管理クラスターの config map 内に kubelet 設定を追加します。

   kubelet 設定を持つ ConfigMap オブジェクトの例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: <configmap_name> 1
  namespace: clusters
data:
  config: |
    apiVersion: machineconfiguration.openshift.io/v1
    kind: KubeletConfig
    metadata:
      name: <kubeletconfig_name> 2
    spec:
      kubeletConfig:
        registerWithTaints:
        - key: "example.sh/unregistered"
          value: "true"
          effect: "NoExecute"
```

1 **<configmap_name>** は、設定マップの名前に置き換えます。

2 **<kubeletconfig_name>** は、**KubeletConfig** リソースの名前に置き換えます。

2. 次のコマンドを入力して、config map をノードプールに適用します。

```
$ oc edit nodepool <nodepool_name> --namespace clusters (1)
```

(1) **<nodepool_name>** をノードプールの名前に置き換えます。

NodePool リソース設定の例

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
# ...
  name: nodepool-1
  namespace: clusters
# ...
spec:
  config:
  - name: <configmap_name> (1)
# ...
```

(1) **<configmap_name>** は、設定マップの名前に置き換えます。

4.3. ホステッドクラスターにおけるノードのチューニング設定

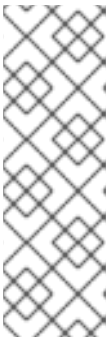
ホステッドクラスター内のノードでノードレベルのチューニングを設定するには、Node Tuning Operator を使用できます。ホストされたコントロールプレーンでは、**Tuned** オブジェクトを含む config map を作成し、ノードプールでそれらの config map を参照することで、ノードのチューニングを設定できます。

手順

1. チューニングされた有効なマニフェストを含む config map 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 config map を参照できます。Hosted Control Plane では、Node Tuning Operator が Tuned CR の名前にノードプール名と namespace のハッシュを追加して、それらを区別します。この場合を除き、同じホステッド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

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** で参照したことで、Node Tuning Operator により **Tuned** オブジェクトがホステッドクラスターに同期されます。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. ホステッドクラスター内の **Tuned** オブジェクトをリスト表示します。

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

出力例

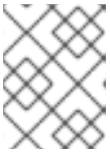
```
NAME       AGE
default    7m36s
rendered   7m36s
tuned-1    65s
```

2. ホステッドクラスター内の **Profile** オブジェクトをリスト表示します。

```
$ 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
```

4.4. HOSTED CONTROL PLANE 用の SR-IOV OPERATOR のデプロイ

重要

AWS プラットフォーム上の Hosted Control Plane は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

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

- テクノロジープレビュー機能のサポート範囲

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: stable
  name: sriov-network-operator
  config:
    nodeSelector:
      node-role.kubernetes.io/worker: ""
  source: 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.4.16.0-202211021237   SR-IOV Network Operator   4.16.0-
202211021237   sriov-network-operator.4.16.0-202210290517   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
```

4.5. ホステッドクラスターの NTP サーバーの設定

Butane を使用して、ホステッドクラスターの Network Time Protocol (NTP) サーバーを設定できます。

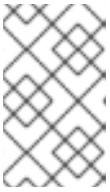
手順

1. **chrony.conf** ファイルの内容を含む Butane 設定ファイル **99-worker-chrony.bu** を作成します。Butane の詳細は、「Butane を使用したマシン設定の作成」を参照してください。

99-worker-chrony.bu の設定例

```
# ...
variant: openshift
version: 4.16.0
metadata:
  name: 99-worker-chrony
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
  - path: /etc/chrony.conf
    mode: 0644 (1)
    overwrite: true
    contents:
      inline: |
        pool 0.rhel.pool.ntp.org iburst (2)
        driftfile /var/lib/chrony/drift
        makestep 1.0 3
        rtcsync
        logdir /var/log/chrony
# ...
```

1. マシン設定ファイルの **mode** フィールドに 8 進数の値でモードを指定します。ファイルを作成して変更を適用すると、**mode** フィールドは 10 進数値に変換されます。
2. Dynamic Host Configuration Protocol (DHCP) サーバーが提供するタイムソースなど、有効で到達可能なタイムソースを指定します。



注記

マシン間通信の場合、User Datagram Protocol (UDP) ポート上の NTP は **123** です。外部 NTP タイムサーバーを設定した場合は、UDP ポート **123** を開く必要があります。

2. Butane を使用して、Butane がノードに送信する設定を含む **MachineConfig** オブジェクトファイル **99-worker-chrony.yaml** を生成します。以下のコマンドを実行します。

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

99-worker-chrony.yaml の設定例

```
# Generated by Butane; do not edit
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: /example/path
```

3. 管理クラスターの config map 内に **99-worker-chrony.yaml** ファイルの内容を追加します。

config map の例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: <configmap_name>
  namespace: <namespace> 1
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: /example/path
# ...
```

1 **<namespace>** は、ノードプールを作成した namespace の名前 (**clusters** など) に置き換えます。

4. 次のコマンドを実行して、config map をノードプールに適用します。

```
$ oc edit nodepool <nodepool_name> --namespace <hosted_cluster_namespace>
```

NodePool の設定例

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
# ...
  name: nodepool-1
  namespace: clusters
# ...
spec:
  config:
  - name: <configmap_name> 1
# ...
```

1 **<configmap_name>** は、設定マップの名前に置き換えます。

5. **InfraEnv** カスタムリソース (CR) を定義する **infra-env.yaml** ファイルに NTP サーバーのリストを追加します。

infra-env.yaml ファイルの例

```
apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
# ...
spec:
  additionalNTPSources:
  - <ntp_server> 1
```

```
  - <ntp_server1>
  - <ntp_server2>
# ...
```

1 **<ntp_server>** は、NTP サーバーの名前に置き換えます。ホストインベントリーと **InfraEnv** CR の作成の詳細は、「ホストインベントリーの作成」を参照してください。

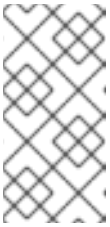
6. 次のコマンドを実行して、**InfraEnv** CR を適用します。

```
$ oc apply -f infra-env.yaml
```

検証

- 次のフィールドを確認し、ホストインベントリーのステータスを確認します。
  - **conditions**: イメージが正常に作成されたかどうかを示す標準の Kubernetes の状態。
  - **isoDownloadURL**: Discovery Image をダウンロードするための URL。
  - **createdTime**: イメージが最後に作成された時刻。**InfraEnv** CR を変更する場合は、新しいイメージをダウンロードする前に、必ずタイムスタンプを更新してください。
    次のコマンドを実行して、ホストインベントリーが作成されたことを確認します。

    ```
    $ oc describe infraenv <infraenv_resource_name> -n <infraenv_namespace>
    ```

    注記

    InfraEnv CR を変更する場合は、**createdTime** フィールドを調べて、**InfraEnv** CR によって新しい Discovery Image が作成されたことを確認してください。すでにホストを起動している場合は、最新の Discovery Image でホストを再起動します。

関連情報

- Butane でのマシン設定の作成
- ホストインベントリーの作成

第5章 ホステッドクラスターでのフィーチャーゲートの使用

ホステッドクラスターでフィーチャーゲートを使用して、デフォルトの機能セットに含まれていない機能を有効にすることができます。ホステッドクラスターでフィーチャーゲートを使用すると、**TechPreviewNoUpgrade** 機能セットを有効にすることができます。

5.1. フィーチャーゲートを使用した機能セットの有効化

OpenShift CLI を使用して **HostedCluster** カスタムリソース (CR) を編集することにより、ホステッドクラスターで **TechPreviewNoUpgrade** 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

1. 次のコマンドを実行して、ホスティングクラスターで **HostedCluster** CR を編集するために開きます。

```
$ oc edit hostedcluster <hosted_cluster_name> -n <hosted_cluster_namespace>
```

2. **featureSet** フィールドに値を入力して機能セットを定義します。以下に例を示します。

```
apiVersion: hypershift.openshift.io/v1beta1
kind: HostedCluster
metadata:
  name: <hosted_cluster_name> 1
  namespace: <hosted_cluster_namespace> 2
spec:
  configuration:
    featureGate:
      featureSet: TechPreviewNoUpgrade 3
```

1. ホステッドクラスターの名前を指定します。
2. ホステッドクラスターの namespace を指定します。
3. 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。

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。この機能セットを使用すると、該当するテクノロジープレビュー機能をテストクラスターで有効にして、完全にテストすることができます。実稼働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。

3. 変更を適用するためにファイルを保存します。

検証

- 次のコマンドを実行して、ホステッドクラスターで **TechPreviewNoUpgrade** フィーチャーゲートが有効になっていることを確認します。

```
$ oc get featuregate cluster -o yaml
```

関連情報

- FeatureGate [config.openshift.io/v1]

第6章 HOSTED CONTROL PLANE の証明書の設定

Hosted Control Plane の場合、証明書の設定手順はスタンドアロンの OpenShift Container Platform の場合とは異なります。

6.1. ホステッドクラスターでカスタム API サーバー証明書を設定する

API サーバーのカスタム証明書を設定するには、**HostedCluster** 設定の **spec.configuration.apiServer** セクションで証明書の詳細を指定します。

カスタム証明書は、Day 1 操作または Day 2 操作の中で設定できます。ただし、サービス公開ストラテジーはホステッドクラスターの作成中に設定した後は変更できないため、設定する予定の Kubernetes API サーバーのホスト名を知っておく必要があります。

前提条件

- 管理クラスターにカスタム証明書が含まれる Kubernetes シークレットを作成しました。シークレットには次の鍵が含まれています。
  - **tls.crt**: 証明書
  - **tls.key**: 秘密鍵
- **HostedCluster** 設定にロードバランサーを使用するサービス公開ストラテジーが含まれている場合は、証明書のサブジェクト代替名 (SAN) が内部 API エンドポイント (**api-int**) と競合しないことを確認してください。内部 API エンドポイントは、プラットフォームによって自動的に作成および管理されます。カスタム証明書と内部 API エンドポイントの両方で同じホスト名を使用すると、ルーティングの競合が発生する可能性があります。このルールの唯一の例外は、**Private** または **PublicAndPrivate** 設定で AWS をプロバイダーとして使用する場合です。このような場合、SAN の競合はプラットフォームによって管理されます。
- 証明書は外部 API エンドポイントに対して有効である必要があります。
- 証明書の有効期間は、クラスターの予想されるライフサイクルと一致します。

手順

1. 次のコマンドを入力して、カスタム証明書を使用してシークレットを作成します。

```
$ oc create secret tls sample-hosted-kas-custom-cert \
  --cert=path/to/cert.crt \
  --key=path/to/key.key \
  -n <hosted_cluster_namespace>
```

2. 次の例に示すように、カスタム証明書の詳細を使用して **HostedCluster** 設定を更新します。

```
spec:
  configuration:
    apiServer:
      servingCerts:
        namedCertificates:
        - names: 1
```

```
    - api-custom-cert-sample-hosted.sample-hosted.example.com
    servingCertificate: 2
      name: sample-hosted-kas-custom-cert
```

1 証明書が有効な DNS の名前のリスト。

2 カスタム証明書が含まれるシークレットの名前。

3. 次のコマンドを入力して、**HostedCluster** 設定に変更を適用します。

```
$ oc apply -f <hosted_cluster_config>.yaml
```

検証

- API サーバー Pod をチェックして、新しい証明書がマウントされていることを確認します。
- カスタムドメイン名を使用して API サーバーへの接続をテストします。
- ブラウザーで、または **openssl** などのツールを使用して、証明書の詳細を確認します。

6.2. ホステッドクラスター用の KUBERNETES API サーバーの設定

ホステッドクラスター用に Kubernetes API サーバーをカスタマイズする場合は、次の手順を実行します。

前提条件

- 実行中のホステッドクラスターがある。
- **HostedCluster** リソースを変更するためのアクセス権がある。
- Kubernetes API サーバーに使用するカスタム DNS ドメインがある。
  - カスタム DNS ドメインは適切に設定され、解決可能である。
  - DNS ドメインには有効な TLS 証明書が設定されている。
  - ドメインへのネットワークアクセスが環境内で適切に設定されている。
  - カスタム DNS ドメインは、ホステッドクラスター全体において一意である。
- カスタム証明書を設定した。詳細は「、ホステッドクラスターでのカスタム API サーバー証明書の設定」を参照してください。

手順

1. プロバイダープラットフォームで、**kubeAPIServerDNSName** URL が Kubernetes API サーバーが公開されている IP アドレスを指すように DNS レコードを設定します。DNS レコードは適切に設定され、クラスターから解決可能である必要があります。

   DNS レコードの設定に使用するコマンドの例

```
$ dig + short kubeAPIServerDNSName
```

2. **HostedCluster** 仕様で、次の例に示すように **kubeAPIServerDNSName** フィールドを変更します。

```
apiVersion: hypershift.openshift.io/v1beta1
kind: HostedCluster
metadata:
  name: <hosted_cluster_name>
  namespace: <hosted_cluster_namespace>
spec:
  configuration:
    apiServer:
      servingCerts:
        namedCertificates:
        - names: 1
          - api-custom-cert-sample-hosted.sample-hosted.example.com
          servingCertificate: 2
            name: sample-hosted-kas-custom-cert
  kubeAPIServerDNSName: api-custom-cert-sample-hosted.sample-hosted.example.com 3
# ...
```

1. 証明書が有効な DNS の名前のリスト。このフィールドにリストされる名前は、**spec.servicePublishingStrategy.*hostname** フィールドに指定された名前と同じにすることはできません。
2. カスタム証明書が含まれるシークレットの名前。
3. このフィールドは、API サーバーのエンドポイントとして使用される URI を受け入れます。

3. 次のコマンドを入力して設定を適用します。

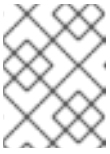
```
$ oc -f <hosted_cluster_spec>.yaml
```

設定が適用されると、HyperShift Operator はカスタム DNS ドメインを指す新しい **kubeconfig** シークレットを生成します。

4. CLI またはコンソールを使用して **kubeconfig** シークレットを取得します。
   a. CLI を使用してシークレットを取得するには、次のコマンドを入力します。

```
$ kubectl get secret <hosted_cluster_name>-custom-admin-kubeconfig \
  -n <cluster_namespace> \
  -o jsonpath='{.data.kubeconfig}' | base64 -d
```

   b. コンソールを使用してシークレットを取得するには、ホステッドクラスターに移動し、**Download Kubeconfig** をクリックします。



注記

コンソールで **show login command** オプションを使用して、新しい **kubeconfig** シークレットを使用することはできません。

6.3. カスタム DNS を使用してホステッドクラスターにアクセスする際のトラブルシューティング

カスタム DNS を使用してホステッドクラスターにアクセスするときに問題が発生した場合は、次のステップを実行します。

手順

1. DNS レコードが適切に設定され、解決されていることを確認します。
2. 次のコマンドを入力して、カスタムドメインの TLS 証明書が有効であること、およびドメインの SAN が正しいことを確認します。

```
$ oc get secret \
  -n clusters <serving_certificate_name> \
  -o jsonpath='{.data.tls\.crt}' | base64 \
  -d |openssl x509 -text -noout -
```

3. カスタムドメインへのネットワーク接続が機能していることを確認します。
4. 次の例に示すように、**HostedCluster** リソースで、ステータスに正しいカスタム **kubeconfig** 情報が表示されていることを確認します。

   HostedCluster ステータスの例

```
status:
  customKubeconfig:
    name: sample-hosted-custom-admin-kubeconfig
```

5. 次のコマンドを入力して、**HostedControlPlane** namespace の **kube-apiserver** ログを確認します。

```
$ oc logs -n <hosted_control_plane_namespace> \
  -l app=kube-apiserver -f -c kube-apiserver
```

第7章 HOSTED CONTROL PLANE の更新

Hosted Control Plane の更新には、ホステッド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。

7.1. HOSTED CONTROL PLANE をアップグレードするための要件

multicluster engine for Kubernetes Operator は、1 つ以上の OpenShift Container Platform クラスターを管理できます。OpenShift Container Platform でホステッドクラスターを作成した後、ホステッドクラスターをマネージドクラスターとして multicluster engine Operator にインポートする必要があります。その後、OpenShift Container Platform クラスターを管理クラスターとして使用できます。

Hosted Control Plane の更新を開始する前に、次の要件を考慮してください。

- OpenShift Virtualization をプロバイダーとして使用する場合は、OpenShift Container Platform クラスターにベアメタルプラットフォームを使用する必要があります。
- ホステッドクラスターのクラウドプラットフォームとして、ベアメタルまたは OpenShift Virtualization を使用する必要があります。ホステッドクラスターのプラットフォームタイプは、**HostedCluster** カスタムリソース (CR) の **spec.Platform.type** 仕様で確認できます。

以下のタスクを完了して、OpenShift Container Platform クラスター、マルチクラスターエンジン Operator、ホストされたクラスター、およびノードプールをアップグレードする必要があります。

1. OpenShift Container Platform クラスターを最新バージョンにアップグレードします。詳細は、「Web コンソールを使用してクラスターを更新」または「CLI を使用したクラスターの更新」を参照してください。
2. multicluster engine Operator を最新バージョンにアップグレードします。詳細は、「インストールされている Operator の更新」を参照してください。
3. ホステッドクラスターとノードプールを以前の OpenShift Container Platform バージョンから最新バージョンにアップグレードします。詳細は、「ホステッドクラスターのコントロールプレーンの更新」および「ホステッドクラスターのノードプールの更新」を参照してください。

関連情報

- Web コンソールを使用してクラスターを更新
- CLI を使用したクラスターの更新
- インストール済み Operators の更新

7.2. ホステッドクラスターのチャネルの設定

利用可能な更新は、**HostedCluster** カスタムリソース (CR) の **HostedCluster.Status** フィールドで確認できます。

利用可能な更新は、ホステッドクラスターの Cluster Version Operator (CVO) からは取得されません。利用可能な更新のリストは、**HostedCluster** カスタムリソース (CR) の次のフィールドに含まれている利用可能な更新とは異なる場合があります。

- **status.version.availableUpdates**

- **status.version.conditionalUpdates**

初期の **HostedCluster** CR には、**status.version.availableUpdates** フィールドと **status.version.conditionalUpdates** フィールドに情報が含まれていません。**spec.channel** フィールドを OpenShift Container Platform の stable リリースバージョンに設定すると、HyperShift Operator が **HostedCluster** CR をリコンサイルし、利用可能な更新と条件付き更新で **status.version** フィールドを更新します。

チャネル設定を含む **HostedCluster** CR の次の例を参照してください。

```
spec:
  autoscaling: {}
  channel: stable-4.y 1
  clusterID: d6d42268-7dff-4d37-92cf-691bd2d42f41
  configuration: {}
  controllerAvailabilityPolicy: SingleReplica
  dns:
    baseDomain: dev11.red-chesterfield.com
    privateZoneID: Z0180092I0DQRKL55LN0
    publicZoneID: Z00206462VG6ZP0H2QLWK
```

1 **<4.y>** は、**spec.release** で指定した OpenShift Container Platform リリースバージョンに置き換えます。たとえば、**spec.release** を **ocp-release:4.16.4-multi** に設定する場合は、**spec.channel** を **stable-4.16** に設定する必要があります。

HostedCluster CR でチャネルを設定した後、**status.version.availableUpdates** フィールドと **status.version.conditionalUpdates** フィールドの出力を表示するには、次のコマンドを実行します。

```
$ oc get -n <hosted_cluster_namespace> hostedcluster <hosted_cluster_name> -o yaml
```

出力例

```
version:
  availableUpdates:
  - channels:
    - candidate-4.16
    - candidate-4.17
    - eus-4.16
    - fast-4.16
    - stable-4.16
    image: quay.io/openshift-release-dev/ocp-release@sha256:b7517d13514c6308ae16c5fd8108133754eb922cd37403ed27c846c129e67a9a
    url: https://access.redhat.com/errata/RHBA-2024:6401
    version: 4.16.11
  - channels:
    - candidate-4.16
    - candidate-4.17
    - eus-4.16
    - fast-4.16
    - stable-4.16
    image: quay.io/openshift-release-dev/ocp-release@sha256:d08e7c8374142c239a07d7b27d1170eae2b0d9f00ccf074c3f13228a1761c162
    url: https://access.redhat.com/errata/RHSA-2024:6004
```

```
    version: 4.16.10
  - channels:
    - candidate-4.16
    - candidate-4.17
    - eus-4.16
    - fast-4.16
    - stable-4.16
    image: quay.io/openshift-release-dev/ocp-release@sha256:6a80ac72a60635a313ae511f0959cc267a21a89c7654f1c15ee16657aafa41a0
    url: https://access.redhat.com/errata/RHBA-2024:5757
    version: 4.16.9
  - channels:
    - candidate-4.16
    - candidate-4.17
    - eus-4.16
    - fast-4.16
    - stable-4.16
    image: quay.io/openshift-release-dev/ocp-release@sha256:ea624ae7d91d3f15094e9e15037244679678bdc89e5a29834b2ddb7e1d9b57e6
    url: https://access.redhat.com/errata/RHSA-2024:5422
    version: 4.16.8
  - channels:
    - candidate-4.16
    - candidate-4.17
    - eus-4.16
    - fast-4.16
    - stable-4.16
    image: quay.io/openshift-release-dev/ocp-release@sha256:e4102eb226130117a0775a83769fe8edb029f0a17b6cbca98a682e3f1225d6b7
    url: https://access.redhat.com/errata/RHSA-2024:4965
    version: 4.16.6
  - channels:
    - candidate-4.16
    - candidate-4.17
    - eus-4.16
    - fast-4.16
    - stable-4.16
    image: quay.io/openshift-release-dev/ocp-release@sha256:f828eda3eaac179e9463ec7b1ed6baeba2cd5bd3f1dd56655796c86260db819b
    url: https://access.redhat.com/errata/RHBA-2024:4855
    version: 4.16.5
  conditionalUpdates:
  - conditions:
    - lastTransitionTime: "2024-09-23T22:33:38Z"
      message: |-
        Could not evaluate exposure to update risk SRIOVFailedToConfigureVF (creating PromQL round-tripper: unable to load specified CA cert /etc/tls/service-ca/service-ca.crt: open /etc/tls/service-ca/service-ca.crt: no such file or directory)
          SRIOVFailedToConfigureVF description: OCP Versions 4.14.34, 4.15.25, 4.16.7 and ALL subsequent versions include kernel datastructure changes which are not compatible with older versions of the SR-IOV operator. Please update SR-IOV operator to versions dated 20240826 or newer before updating OCP.
          SRIOVFailedToConfigureVF URL: https://issues.redhat.com/browse/NHE-1171
      reason: EvaluationFailed
      status: Unknown
      type: Recommended
```

```
    release:
      channels:
      - candidate-4.16
      - candidate-4.17
      - eus-4.16
      - fast-4.16
      - stable-4.16
      image: quay.io/openshift-release-dev/ocp-
release@sha256:fb321a3f50596b43704dbbed2e51fdefd7a7fd488ee99655d03784d0cd02283f
      url: https://access.redhat.com/errata/RHSA-2024:5107
      version: 4.16.7
    risks:
    - matchingRules:
      - promql:
          promql: |
            group(csv_succeeded{_id="d6d42268-7dff-4d37-92cf-691bd2d42f41", name=~"sriov-network-
operator[.].*"})
            or
            0 * group(csv_count{_id="d6d42268-7dff-4d37-92cf-691bd2d42f41"})
        type: PromQL
      message: OCP Versions 4.14.34, 4.15.25, 4.16.7 and ALL subsequent versions
        include kernel datastructure changes which are not compatible with older
        versions of the SR-IOV operator. Please update SR-IOV operator to versions
        dated 20240826 or newer before updating OCP.
      name: SRIOVFailedToConfigureVF
      url: https://issues.redhat.com/browse/NHE-1171
```

7.3. ホステッドクラスターの OPENSHIFT CONTAINER PLATFORM バージョンの更新

Hosted Control Plane は、コントロールプレーンとデータプレーン間の更新の分離を可能にします。

クラスターサービスプロバイダーやクラスター管理者は、コントロールプレーンとデータを個別に管理できます。

コントロールプレーンは **HostedCluster** カスタムリソース (CR) を変更することで更新でき、ノードは **NodePool** CR を変更することで更新できます。**HostedCluster** CR と **NodePool** CR では、どちらも **.release** フィールドで OpenShift Container Platform リリースイメージを指定します。

更新プロセス中にホステッドクラスターを完全に機能させ続けるには、コントロールプレーンとノードの更新が Kubernetes バージョンスキューポリシー に準拠している必要があります。

7.3.1. multicluster engine Operator ハブ管理クラスター

multicluster engine for Kubernetes Operator には、管理クラスターのサポート対象状態を維持するために、特定の OpenShift Container Platform バージョンが必要です。multicluster engine Operator は、OpenShift Container Platform Web コンソールの OperatorHub からインストールできます。

multicluster engine Operator のバージョンは、次のサポートマトリックスを参照してください。

- multicluster engine Operator 2.7
- multicluster engine Operator 2.6
- multicluster engine Operator 2.5

- multicluster engine Operator 2.4

multicluster engine Operator は、次の OpenShift Container Platform バージョンをサポートしています。

- 最新の未リリースバージョン
- 最新リリースバージョン
- 最新リリースバージョンの 2 つ前のバージョン

Red Hat Advanced Cluster Management (RHACM) の一部として multicluster engine Operator バージョンを入手することもできます。

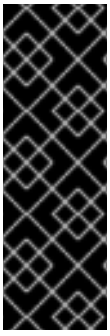
7.3.2. ホステッドクラスターでサポートされる OpenShift Container Platform のバージョン

ホステッドクラスターをデプロイする場合、管理クラスターの OpenShift Container Platform バージョンは、ホステッドクラスターの OpenShift Container Platform バージョンに影響しません。

HyperShift Operator は、**hypershift** namespace に **supported-versions** ConfigMap を作成します。**supported-versions** ConfigMap には、デプロイ可能なサポートされている OpenShift Container Platform バージョンの範囲が記述されています。

supported-versions ConfigMap の次の例を参照してください。

```
apiVersion: v1
data:
  server-version: 2f6cfe21a0861dea3130f3bed0d3ae5553b8c28b
  supported-versions: '{"versions":["4.17","4.16","4.15","4.14"]}'
kind: ConfigMap
metadata:
  creationTimestamp: "2024-06-20T07:12:31Z"
  labels:
    hypershift.openshift.io/supported-versions: "true"
  name: supported-versions
  namespace: hypershift
  resourceVersion: "927029"
  uid: f6336f91-33d3-472d-b747-94abae725f70
```



重要

ホステッドクラスターを作成するには、サポートバージョン範囲内の OpenShift Container Platform バージョンを使用する必要があります。ただし、multicluster engine Operator が管理できるのは、**n+1** から **n-2** までの OpenShift Container Platform バージョンのみです。ここで **n** は現在のマイナーバージョンを定義します。multicluster engine Operator サポートマトリックスをチェックすると、multicluster engine Operator によって管理されるホステッドクラスターが、サポートされている OpenShift Container Platform の範囲内であることを確認できます。

上位バージョンのホステッドクラスターを OpenShift Container Platform にデプロイするには、multicluster engine Operator を新しいマイナーバージョンリリースに更新して、新しいバージョンの Hypershift Operator をデプロイする必要があります。multicluster engine Operator を新しいパッチ (z-stream) にアップグレードしても、HyperShift Operator は次のバージョンに更新されません。

hcp version コマンドの次の出力例を参照してください。管理クラスターの OpenShift Container Platform 4.16 でサポートされている OpenShift Container Platform バージョンを示しています。

```
Client Version: openshift/hypershift: fe67b47fb60e483fe60e4755a02b3be393256343. Latest supported OCP: 4.17.0
Server Version: 05864f61f24a8517731664f8091cedcfc5f9b60d
Server Supports OCP Versions: 4.17, 4.16, 4.15, 4.14
```

7.4. ホステッドクラスターの更新

spec.release.image 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.image** 値を **HostedControlPlane.spec.releaseImage** 値に送信し、適切な Control Plane Operator のバージョンを実行します。

Hosted Control Plane は、新しいバージョンの Cluster Version Operator (CVO) により、新しいバージョンのコントロールプレーンコンポーネントと OpenShift Container Platform コンポーネントのロールアウトを管理します。

重要

Hosted Control Plane では、**NodeHealthCheck** リソースが CVO のステータスを検出できません。クラスター管理者は、クラスターの更新などの重要な操作を実行する前に、新しい修復アクションがクラスターの更新に干渉するのを防ぐために、**NodeHealthCheck** によってトリガーされた修復を手動で一時停止する必要があります。

修復を一時停止するには、**NodeHealthCheck** リソースの **pauseRequests** フィールドの値として、文字列の配列 (例: **pause-test-cluster**) を入力します。詳細は、Node Health Check Operator について を参照してください。

クラスターの更新が完了したら、修復を編集または削除できます。**Compute** → **NodeHealthCheck** ページに移動し、ノードヘルスチェックをクリックして、**Actions** をクリックすると、ドロップダウンリストが表示されます。

7.5. 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、replace 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。

7.5.1. ノードプールの replace 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。

replace 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。

7.5.2. 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。

7.6. ホステッドクラスターのノードプールの更新

ホステッドクラスターのノードプールを更新することで、OpenShift Container Platform のバージョンを更新できます。ノードプールのバージョンが Hosted Control Plane のバージョンを超えることはできません。

NodePool カスタムリソース (CR) の **.spec.release** フィールドは、ノードプールのバージョンを示します。

手順

- 次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 patch nodepool <node_pool_name> -n <hosted_cluster_namespace> --type=merge -p
'{"spec":{"nodeDrainTimeout":"60s","release":{"image":"<openshift_release_image>"}}}' 1
2
```

1. **<node_pool_name>** と **<hosted_cluster_namespace>** を、それぞれノードプール名とホステッドクラスターの namespace に置き換えます。
2. **<openshift_release_image>** 変数は、アップグレードする新しい OpenShift Container Platform リリースイメージを指定します (例: **quay.io/openshift-release-dev/ocp-release:4.y.z-x86_64**)。**<4.y.z>** をサポートされている OpenShift Container Platform バージョンに置き換えます。

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、次のコマンドを実行して、ノードプールの **.status.conditions** 値を確認します。

```
$ oc get -n <hosted_cluster_namespace> nodepool <node_pool_name> -o yaml
```

出力例

```
status:
 conditions:
 - lastTransitionTime: "2024-05-20T15:00:40Z"
     message: 'Using release image: quay.io/openshift-release-dev/ocp-release:4.y.z-x86_64'
(1)
     reason: AsExpected
     status: "True"
     type: ValidReleaseImage
```

1. **<4.y.z>** をサポートされている OpenShift Container Platform バージョンに置き換えます。

7.7. ホステッドクラスターのコントロールプレーンの更新

Hosted Control Plane でホステッドクラスターを更新することで、OpenShift Container Platform のバージョンをアップグレードできます。**HostedCluster** カスタムリソース (CR) の **.spec.release** は、コントロールプレーンのバージョンを示します。**HostedCluster** は、**.spec.release** フィールドを **HostedControlPlane.spec.release** に更新し、適切な Control Plane Operator のバージョンを実行します。

HostedControlPlane リソースは、新しいバージョンの Cluster Version Operator (CVO) により、データプレーン内の OpenShift Container Platform コンポーネントとともに、新しいバージョンのコントロールプレーンコンポーネントのロールアウトを調整します。**HostedControlPlane** には次のアーティファクトが含まれています。

- CVO
- Cluster Network Operator (CNO)
- Cluster Ingress Operator
- Kube API サーバー、スケジューラー、およびマネージャーのマニフェスト
- Machine approver
- Autoscaler
- Kube API サーバー、Ignition、Konnectivity など、コントロールプレーンエンドポイントの Ingress を有効にするインフラストラクチャーリソース

HostedCluster CR の **.spec.release** フィールドを設定すると、**status.version.availableUpdates** フィールドと **status.version.conditionalUpdates** フィールドの情報を使用してコントロールプレーンを更新できます。

手順

1. 次のコマンドを実行して、ホステッドクラスターに **hypershift.openshift.io/force-upgrade-to=<openshift_release_image>** アノテーションを追加します。

```
$ oc annotate hostedcluster -n <hosted_cluster_namespace> <hosted_cluster_name>
"hypershift.openshift.io/force-upgrade-to=<openshift_release_image>" --overwrite (1) (2)
```

1. **<hosted_cluster_name>** と **<hosted_cluster_namespace>** を、それぞれホステッドクラスター名とホステッドクラスター namespace に置き換えます。
2. **<openshift_release_image>** 変数は、アップグレードする新しい OpenShift Container Platform リリースイメージを指定します (例: **quay.io/openshift-release-dev/ocp-release:4.y.z-x86_64**)。**<4.y.z>** をサポートされている OpenShift Container Platform バージョンに置き換えます。

2. 次のコマンドを実行して、ホステッドクラスターの **spec.release.image** 値を変更します。

```
$ oc patch hostedcluster <hosted_cluster_name> -n <hosted_cluster_namespace> --type=merge -p '{"spec":{"release":{"image":"<openshift_release_image>"}}}'
```

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、次のコマンドを実行して、ホステッドクラスターの **.status.conditions** と **.status.version** の値を確認します。

```
$ oc get -n <hosted_cluster_namespace> hostedcluster <hosted_cluster_name> -o yaml
```

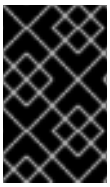
出力例

```
status:
 conditions:
 - lastTransitionTime: "2024-05-20T15:01:01Z"
       message: Payload loaded version="4.y.z" image="quay.io/openshift-release-dev/ocp-release:4.y.z-x86_64" (1)
       status: "True"
       type: ClusterVersionReleaseAccepted
#...
version:
     availableUpdates: null
     desired:
     image: quay.io/openshift-release-dev/ocp-release:4.y.z-x86_64 (2)
     version: 4.y.z
```

1 2 **<4.y.z>** をサポートされている OpenShift Container Platform バージョンに置き換えます。

7.8. MULTICLUSTER ENGINE OPERATOR のコンソールを使用したホステッドクラスターの更新

multicluster engine Operator のコンソールを使用して、ホステッドクラスターを更新できます。



重要

ホステッドクラスターを更新する前に、ホステッドクラスターの利用可能な更新と条件付き更新を参照する必要があります。間違ったリリースバージョンを選択すると、ホステッドクラスターが壊れる可能性があります。

手順

1. **All clusters** を選択します。
2. **Infrastructure → Clusters** に移動して、管理対象のホステッドクラスターを表示します。
3. **Upgrade available** リンクをクリックして、コントロールプレーンとノードプールを更新します。

7.9. インポートされたホステッドクラスターの管理の制限

ホステッドクラスターは、スタンドアロンの OpenShift Container Platform やサードパーティーのクラスターとは異なり、multicluster engine for Kubernetes Operator に自動的にインポートされます。ホステッドクラスターは、エージェントがクラスターのリソースを使用しないように、一部のエージェントを **ホステッドモード** で実行します。

ホステッドクラスターを自動的にインポートした場合は、管理クラスターの **HostedCluster** リソースを使用して、ホステッドクラスター内のノードプールとコントロールプレーンを更新できます。ノードプールとコントロールプレーンを更新するには、「ホステッドクラスターのノードプールの更新」と「ホステッドクラスターのコントロールプレーンの更新」を参照してください。

Red Hat Advanced Cluster Management (RHACM) を使用すると、ホステッドクラスターをローカルの multicluster engine Operator 以外の場所にインポートできます。詳細は、「Red Hat Advanced Cluster Management での multicluster engine for Kubernetes Operator ホステッドクラスターの検出」を参照してください。

このトポロジーでは、クラスターがホストされている multicluster engine for Kubernetes Operator のコマンドラインインターフェイスまたはコンソールを使用して、ホステッドクラスターを更新する必要があります。RHACM ハブクラスターを介してホステッドクラスターを更新することはできません。

関連情報

- ホステッドクラスターのノードプールの更新
- ホステッドクラスターのコントロールプレーンの更新
- Red Hat Advanced Cluster Management での multicluster engine for Kubernetes Operator ホステッドクラスターの検出

第8章 HOSTED CONTROL PLANE の可観測性

メトリクスセットを設定することで、Hosted Control Plane のメトリクスを収集できます。HyperShift Operator は、管理対象のホステッドクラスターごとに、管理クラスター内のモニタリングダッシュボードを作成または削除できます。

8.1. HOSTED CONTROL PLANE のメトリクスセットの設定

Red Hat OpenShift Container Platform の Hosted Control Plane は、各コントロールプレーンの namespace に **ServiceMonitor** リソースを作成します。これにより、Prometheus スタックがコントロールプレーンからメトリクスを収集できるようになります。**ServiceMonitor** リソースは、メトリクスの再ラベル付けを使用して、etcd や Kubernetes API サーバーなどの特定のコンポーネントにどのメトリクスを含めるか、または除外するかを定義します。コントロールプレーンによって生成されるメトリクスの数は、それらを収集する監視スタックのリソース要件に直接影響します。

すべての状況に適用される固定数のメトリクスを生成する代わりに、コントロールプレーンごとに生成するメトリクスのセットを識別するメトリクスセットを設定できます。次のメトリクスセットがサポートされています。

- **Telemetry**: このメトリクスはテレメトリーに必要です。このセットはデフォルトのセットであり、メトリクスの最小セットです。
- **SRE**: このセットには、アラートを生成し、コントロールプレーンコンポーネントのトラブルシューティングを可能にするために必要なメトリクスが含まれています。
- **All**: このセットには、スタンドアロンの OpenShift Container Platform コントロールプレーンコンポーネントによって生成されるすべてのメトリクスが含まれます。

メトリクスセットを設定するには、次のコマンドを入力して、HyperShift Operator デプロイメントで **METRICS_SET** 環境変数を設定します。

```
$ oc set env -n hypershift deployment/operator METRICS_SET=All
```

8.1.1. SRE メトリクスセットの設定

SRE メトリクスセットを指定すると、HyperShift Operator は、単一キー **config** を持つ **sre-metric-set** という名前の config map を検索します。**config** キーの値には、コントロールプレーンコンポーネント別に整理された **RelabelConfigs** のセットが含まれている必要があります。

次のコンポーネントを指定できます。

- **etcd**
- **kubeAPIServer**
- **kubeControllerManager**
- **openshiftAPIServer**
- **openshiftControllerManager**
- **openshiftRouteControllerManager**
- **cvo**

- **olm**
- **catalogOperator**
- **registryOperator**
- **nodeTuningOperator**
- **controlPlaneOperator**
- **hostedClusterConfigOperator**

SRE メトリクスセットの設定を次の例に示します。

```
kubeAPIServer:
  - action:       "drop"
    regex:        "etcd_(debugging|disk|server).*"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:        "apiserver_admission_controller_admission_latencies_seconds_.*"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:        "apiserver_admission_step_admission_latencies_seconds_.*"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:
"scheduler_(e2e_scheduling_latency_microseconds|scheduling_algorithm_predicate_evaluation|schedu
ng_algorithm_priority_evaluation|scheduling_algorithm_preemption_evaluation|scheduling_algorithm_la
ency_microseconds|binding_latency_microseconds|scheduling_latency_seconds)"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:
"apiserver_(request_count|request_latencies|request_latencies_summary|dropped_requests|storage_d
ata_key_generation_latencies_microseconds|storage_transformation_failures_total|storage_transformat
ion_latencies_microseconds|proxy_tunnel_sync_latency_secs)"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:
"docker_(operations|operations_latency_microseconds|operations_errors|operations_timeout)"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:
"reflector_(items_per_list|items_per_watch|list_duration_seconds|lists_total|short_watches_total|watch_
duration_seconds|watches_total)"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:
"etcd_(helper_cache_hit_count|helper_cache_miss_count|helper_cache_entry_count|request_cache_ge
t_latencies_summary|request_cache_add_latencies_summary|request_latencies_summary)"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:        "transformation_(transformation_latencies_microseconds|failures_total)"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:
"network_plugin_operations_latency_microseconds|sync_proxy_rules_latency_microseconds|rest_client
```

```
_request_latency_seconds"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:        "apiserver_request_duration_seconds_bucket;
(0.15|0.25|0.3|0.35|0.4|0.45|0.6|0.7|0.8|0.9|1.25|1.5|1.75|2.5|3|3.5|4.5|6|7|8|9|15|25|30|50)"
    sourceLabels: ["__name__", "le"]
kubeControllerManager:
  - action:       "drop"
    regex:        "etcd_(debugging|disk|request|server).*"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:        "rest_client_request_latency_seconds_(bucket|count|sum)"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:        "root_ca_cert_publisher_sync_duration_seconds_(bucket|count|sum)"
    sourceLabels: ["__name__"]
openshiftAPIServer:
  - action:       "drop"
    regex:        "etcd_(debugging|disk|server).*"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:        "apiserver_admission_controller_admission_latencies_seconds_.*"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:        "apiserver_admission_step_admission_latencies_seconds_.*"
    sourceLabels: ["__name__"]
  - action:       "drop"
    regex:        "apiserver_request_duration_seconds_bucket;
(0.15|0.25|0.3|0.35|0.4|0.45|0.6|0.7|0.8|0.9|1.25|1.5|1.75|2.5|3|3.5|4.5|6|7|8|9|15|25|30|50)"
    sourceLabels: ["__name__", "le"]
openshiftControllerManager:
  - action:       "drop"
    regex:        "etcd_(debugging|disk|request|server).*"
    sourceLabels: ["__name__"]
openshiftRouteControllerManager:
  - action:       "drop"
    regex:        "etcd_(debugging|disk|request|server).*"
    sourceLabels: ["__name__"]
olm:
  - action:       "drop"
    regex:        "etcd_(debugging|disk|server).*"
    sourceLabels: ["__name__"]
catalogOperator:
  - action:       "drop"
    regex:        "etcd_(debugging|disk|server).*"
    sourceLabels: ["__name__"]
cvo:
  - action: drop
    regex: "etcd_(debugging|disk|server).*"
    sourceLabels: ["__name__"]
```

8.2. ホステッドクラスターのモニタリングダッシュボードの有効化

config map を作成することにより、ホステッドクラスターでモニタリングダッシュボードを有効にできます。

手順

1. **local-cluster** namespace に、**hypershift-operator-install-flags** config map を作成します。次の設定例を参照してください。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: hypershift-operator-install-flags
  namespace: local-cluster
data:
  installFlagsToAdd: "--monitoring-dashboards --metrics-set=All" 1
  installFlagsToRemove: ""
```

1 **--monitoring-dashboards --metrics-set=All** フラグは、すべてのメトリクスのモニタリングダッシュボードを追加します。

2. **hypershift** namespace の HyperShift Operator デプロイメントが更新され、次の環境変数が含まれるまで数分待ちます。

```
- name: MONITORING_DASHBOARDS
  value: "1"
```

モニタリングダッシュボードが有効になっている場合、HyperShift Operator が管理するホステッドクラスターごとに、HyperShift Operator が **openshift-config-managed** namespace に **cp-<hosted_cluster_namespace>-<hosted_cluster_name>** という名前の config map を作成します。**<hosted_cluster_namespace>** はホステッドクラスターの namespace、**<hosted_cluster_name>** はホステッドクラスターの名前です。その結果、管理クラスターの管理コンソールに新しいダッシュボードが追加されます。

3. ダッシュボードを表示するには、管理クラスターのコンソールにログインし、**Observe → Dashboards** をクリックして、ホステッドクラスターのダッシュボードに移動します。

4. オプション: ホステッドクラスターのモニタリングダッシュボードを無効にするには、**hypershift-operator-install-flags** config map から **--monitoring-dashboards --metrics-set=All** フラグを削除します。ホステッドクラスターを削除すると、対応するダッシュボードも削除されます。

8.2.1. ダッシュボードのカスタマイズ

各ホステッドクラスターのダッシュボードを生成するために、HyperShift Operator は、Operator の namespace (**hypershift**) の **monitoring-dashboard-template** config map に保存されているテンプレートを使用します。このテンプレートには、ダッシュボードのメトリクスを含む一連の Grafana パネルが含まれています。config map の内容を編集して、ダッシュボードをカスタマイズできます。

ダッシュボードが生成されると、次の文字列が特定のホステッドクラスターに対応する値に置き換えられます。

名前	説明
__NAME__	ホステッドクラスターの名前

名前	説明
__NAMESPACE__	ホステッドクラスターの namespace
__CONTROL_PLANE_NAMESPACE__	ホステッドクラスターのコントロールプレーン Pod が配置される namespace
__CLUSTER_ID__	ホステッドクラスターの UUID。これはホステッドクラスターのメトリクスの **_id** ラベルと一致します。

第9章 HOSTED CONTROL PLANE の高可用性

9.1. HOSTED CONTROL PLANE の高可用性について

次のアクションを実装することで、Hosted Control Plane の高可用性 (HA) を維持できます。

- ホステッドクラスターの etcd メンバーを回復します。
- ホステッドクラスターの etcd をバックアップおよび復元します。
- ホステッドクラスターの障害復旧プロセスを実行します。

9.1.1. 管理クラスターコンポーネントの障害の影響

管理クラスターコンポーネントに障害が発生しても、ワークロードは影響を受けません。OpenShift Container Platform 管理クラスターでは、回復力を提供するために、コントロールプレーンがデータプレーンから分離されています。

次の表は、管理クラスターコンポーネントの障害がコントロールプレーンとデータプレーンに与える影響を示しています。ただし、この表は管理クラスターコンポーネントの障害のすべてのシナリオを網羅しているわけではありません。

表9.1 故障したコンポーネントが Hosted Control Plane に与える影響

故障したコンポーネントの名前	Hosted Control Plane API ステータス	ホステッドクラスターデータプレーンのステータス
ワーカーノード	利用可能	利用可能
アベイラビリティーゾーン	利用可能	利用可能
管理クラスターの制御プレーン	利用可能	利用可能
管理クラスターのコントロールプレーンとワーカーノード	利用不可	利用可能

9.2. 不健全な ETCD クラスターの回復

高可用性コントロールプレーンでは、3 つの etcd Pod が etcd クラスター内のステートフルセットの一部として実行されます。etcd クラスターを回復するには、etcd クラスターの健全性をチェックして、正常でない etcd Pod を特定します。

9.2.1. etcd クラスターのステータスの確認

任意の etcd Pod にログインすると、etcd クラスターの健全性ステータスを確認できます。

手順

1. 次のコマンドを入力して、etcd Pod にログインします。

```
$ oc rsh -n openshift-etcd -c etcd <etcd_pod_name>
```

2. 次のコマンドを入力して、etcd クラスターの健全性ステータスを出力します。

```
sh-4.4# etcdctl endpoint status -w table
```

出力例

```
+----------------------------+-----------------+---------+---------+-----------+-----------+-----------+--
----------+------------------+--------+
|         ENDPOINT           |       ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+----------------------------+-----------------+---------+---------+-----------+-----------+-----------+--
----------+------------------+--------+
| https://192.168.1xxx.20:2379 | 8fxxxxxxxxxx    |  3.5.12 |  123 MB |     false |      false |
10 |     180156 |             180156 |        |
| https://192.168.1xxx.21:2379 | a5xxxxxxxxxx    |  3.5.12 |  122 MB |     false |      false |
10 |     180156 |             180156 |        |
| https://192.168.1xxx.22:2379 | 7cxxxxxxxxxx    |  3.5.12 |  124 MB |      true |      false |
10 |     180156 |             180156 |        |
+----------------------------+-----------------+---------+---------+-----------+-----------+-----------+--
----------+------------------+--------+
```

9.2.2. 障害が発生した etcd Pod の回復

3 ノードクラスターの各 etcd Pod には、データを保存するための独自の永続ボリューム要求 (PVC) があります。データが破損しているか欠落しているために、etcd Pod が失敗する可能性があります。障害が発生した etcd Pod とその PVC を回復できます。

手順

1. etcd Pod が失敗していることを確認するには、次のコマンドを入力します。

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

出力例

```
NAME     READY   STATUS             RESTARTS     AGE
etcd-0   2/2     Running            0            64m
etcd-1   2/2     Running            0            45m
etcd-2   1/2     CrashLoopBackOff   1 (5s ago)   64m
```

失敗した etcd Pod のステータスは **CrashLoopBackOff** または **Error** である可能性があります。

2. 次のコマンドを入力して、障害が発生した Pod とその PVC を削除します。

```
$ oc delete pods etcd-2 -n openshift-etcd
```

検証

- 次のコマンドを実行して、新しい etcd Pod が起動して実行していることを確認します。

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

出力例

```
NAME    READY   STATUS    RESTARTS   AGE
etcd-0  2/2     Running   0          67m
etcd-1  2/2     Running   0          48m
etcd-2  2/2     Running   0          2m2s
```

9.3. オンプレミス環境での ETCD のバックアップと復元

オンプレミス環境のホステッドクラスターで etcd をバックアップおよび復元して、障害を修正できます。

9.3.1. オンプレミス環境のホステッドクラスターでの etcd のバックアップと復元

ホステッドクラスターで etcd をバックアップおよび復元することで、3 ノードクラスターの etcd メンバー内にあるデータの破損や欠落などの障害を修正できます。etcd クラスターの複数メンバーでデータの損失や **CrashLoopBackOff** ステータスが発生する場合、このアプローチにより etcd クォーラムの損失を防ぐことができます。

重要

この手順には、API のダウンタイムが必要です。

前提条件

- **oc** および **jq** バイナリーがインストールされている。

手順

1. まず環境変数を設定し、API サーバーをスケールダウンします。
   a. 必要に応じて値を置き換えて次のコマンドを入力し、ホストされたクラスターの環境変数を設定します。

      ```
      $ CLUSTER_NAME=my-cluster
      ```

      ```
      $ HOSTED_CLUSTER_NAMESPACE=clusters
      ```

      ```
      $ CONTROL_PLANE_NAMESPACE="${HOSTED_CLUSTER_NAMESPACE}-
      ${CLUSTER_NAME}"
      ```

   b. 必要に応じて値を置き換えて次のコマンドを入力し、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 ${HOSTED_CLUSTER_NAMESPACE}
      hostedclusters/${CLUSTER_NAME} -p '{"spec":{"pausedUntil":"true"}}' --type=merge
      ```

   c. 次のコマンドを入力して、API サーバーをスケールダウンします。
      i. **kube-apiserver** をスケールダウンします。

```
$ oc scale -n ${CONTROL_PLANE_NAMESPACE} deployment/kube-apiserver --
replicas=0
```

ii. **openshift-apiserver** をスケールダウンします。

```
$ oc scale -n ${CONTROL_PLANE_NAMESPACE} deployment/openshift-apiserver -
-replicas=0
```

iii. **openshift-oauth-apiserver** をスケールダウンします。

```
$ oc scale -n ${CONTROL_PLANE_NAMESPACE} deployment/openshift-oauth-
apiserver --replicas=0
```

2. 次に、次のいずれかの方法を使用して etcd のスナップショットを取得します。

a. 以前にバックアップした etcd のスナップショットを使用します。

b. 使用可能な etcd Pod がある場合は、次の手順を実行して、アクティブな etcd Pod からスナップショットを取得します。

i. 次のコマンドを入力して、etcd Pod をリスト表示します。

```
$ oc get -n ${CONTROL_PLANE_NAMESPACE} pods -l app=etcd
```

ii. 次のコマンドを入力して、Pod データベースのスナップショットを取得し、マシンのローカルに保存します。

```
$ ETCD_POD=etcd-0
```

```
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} -c etcd -t ${ETCD_POD} -- env
ETCDCTL_API=3 /usr/bin/etcdctl \
--cacert /etc/etcd/tls/etcd-ca/ca.crt \
--cert /etc/etcd/tls/client/etcd-client.crt \
--key /etc/etcd/tls/client/etcd-client.key \
--endpoints=https://localhost:2379 \
snapshot save /var/lib/snapshot.db
```

iii. 次のコマンドを入力して、スナップショットが成功したことを確認します。

```
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} -c etcd -t ${ETCD_POD} -- env
ETCDCTL_API=3 /usr/bin/etcdctl -w table snapshot status /var/lib/snapshot.db
```

c. 次のコマンドを入力して、スナップショットのローカルコピーを作成します。

```
$ oc cp -c etcd
${CONTROL_PLANE_NAMESPACE}/${ETCD_POD}:/var/lib/snapshot.db
/tmp/etcd.snapshot.db
```

i. etcd 永続ストレージからスナップショットデータベースのコピーを作成します。

A. 次のコマンドを入力して、etcd Pod をリスト表示します。

```
$ oc get -n ${CONTROL_PLANE_NAMESPACE} pods -l app=etcd
```

B. 実行中の Pod を検索し、その名前を **ETCD_POD: ETCD_POD=etcd-0** の値として設定し、次のコマンドを入力してそのスナップショットデータベースをコピーします。

```
$ oc cp -c etcd
${CONTROL_PLANE_NAMESPACE}/${ETCD_POD}:/var/lib/data/member/snap/
db /tmp/etcd.snapshot.db
```

3. 次のコマンドを入力して、etcd statefulset をスケールダウンします。

```
$ oc scale -n ${CONTROL_PLANE_NAMESPACE} statefulset/etcd --replicas=0
```

a. 次のコマンドを入力して、2 番目と 3 番目のメンバーのボリュームを削除します。

```
$ oc delete -n ${CONTROL_PLANE_NAMESPACE} pvc/data-etcd-1 pvc/data-etcd-2
```

b. 最初の etcd メンバーのデータにアクセスする Pod を作成します。

i. 次のコマンドを入力して、etcd イメージを取得します。

```
$ ETCD_IMAGE=$(oc get -n ${CONTROL_PLANE_NAMESPACE} statefulset/etcd -
o jsonpath='{ .spec.template.spec.containers[0].image }')
```

ii. etcd データへのアクセスを許可する Pod を作成します。

```
$ cat << EOF | oc apply -n ${CONTROL_PLANE_NAMESPACE} -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: etcd-data
spec:
  replicas: 1
  selector:
    matchLabels:
      app: etcd-data
  template:
    metadata:
      labels:
        app: etcd-data
    spec:
      containers:
      - name: access
        image: $ETCD_IMAGE
        volumeMounts:
        - name: data
          mountPath: /var/lib
        command:
        - /usr/bin/bash
        args:
        - -c
        - |-
          while true; do
            sleep 1000
          done
      volumes:
```

```
    - name: data
      persistentVolumeClaim:
        claimName: data-etcd-0
EOF
```

iii. 次のコマンドを入力して、**etcd-data** Pod のステータスを確認し、実行されるまで待ちます。

```
$ oc get -n ${CONTROL_PLANE_NAMESPACE} pods -l app=etcd-data
```

iv. 次のコマンドを入力して、**etcd-data** Pod の名前を取得します。

```
$ DATA_POD=$(oc get -n ${CONTROL_PLANE_NAMESPACE} pods --no-headers
-l app=etcd-data -o name | cut -d/ -f2)
```

c. 次のコマンドを入力して、etcd スナップショットを Pod にコピーします。

```
$ oc cp /tmp/etcd.snapshot.db
${CONTROL_PLANE_NAMESPACE}/${DATA_POD}:/var/lib/restored.snap.db
```

d. 次のコマンドを入力して、**etcd-data** Pod から古いデータを削除します。

```
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} ${DATA_POD} -- rm -rf /var/lib/data
```

```
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} ${DATA_POD} -- mkdir -p
/var/lib/data
```

e. 次のコマンドを入力して、etcd スナップショットを復元します。

```
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} ${DATA_POD} -- etcdutl snapshot
restore /var/lib/restored.snap.db \
    --data-dir=/var/lib/data --skip-hash-check \
    --name etcd-0 \
    --initial-cluster-token=etcd-cluster \
    --initial-cluster etcd-0=https://etcd-0.etcd-
discovery.${CONTROL_PLANE_NAMESPACE}.svc:2380,etcd-1=https://etcd-1.etcd-
discovery.${CONTROL_PLANE_NAMESPACE}.svc:2380,etcd-2=https://etcd-2.etcd-
discovery.${CONTROL_PLANE_NAMESPACE}.svc:2380 \
    --initial-advertise-peer-urls https://etcd-0.etcd-
discovery.${CONTROL_PLANE_NAMESPACE}.svc:2380
```

f. 次のコマンドを入力して、Pod から一時的な etcd スナップショットを削除します。

```
$ oc exec -n ${CONTROL_PLANE_NAMESPACE} ${DATA_POD} -- rm
/var/lib/restored.snap.db
```

g. 次のコマンドを入力して、データアクセスデプロイメントを削除します。

```
$ oc delete -n ${CONTROL_PLANE_NAMESPACE} deployment/etcd-data
```

h. 次のコマンドを入力して、etcd クラスターをスケールアップします。

```
$ oc scale -n ${CONTROL_PLANE_NAMESPACE} statefulset/etcd --replicas=3
```

i. 次のコマンドを入力して、etcd メンバー Pod が返され、使用可能であると報告されるのを待ちます。

```
$ oc get -n ${CONTROL_PLANE_NAMESPACE} pods -l app=etcd -w
```

j. 次のコマンドを入力して、すべての etcd-writer デプロイメントをスケールアップします。

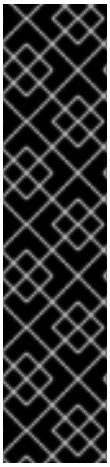
```
$ oc scale deployment -n ${CONTROL_PLANE_NAMESPACE} --replicas=3 kube-apiserver openshift-apiserver openshift-oauth-apiserver
```

4. 次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復元します。

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

9.4. AWS での ETCD のバックアップと復元

障害を修正するために、Amazon Web Services (AWS) でホストされているクラスターで etcd をバックアップおよび復元できます。



重要

AWS プラットフォーム上の Hosted Control Plane は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

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

- テクノロジープレビュー機能のサポート範囲

9.4.1. ホストされたクラスターの etcd のスナップショットを取得

ホステッドクラスターの etcd をバックアップするには、etcd のスナップショットを作成する必要があります。後で、スナップショットを使用して etcd を復元できます。

重要

この手順には、API のダウンタイムが必要です。

手順

1. 次のコマンドを入力して、ホステッドクラスターのリコンシリエーションを一時停止します。

```
$ oc patch -n clusters hostedclusters/<hosted_cluster_name> -p '{"spec":{"pausedUntil":"true"}}' --type=merge
```

2. 次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. etcd スナップショットを取得するには、次のコマンドを実行して、各 etcd コンテナーで **exec** コマンドを使用します。

```
$ oc exec -it <etcd_pod_name> -n <hosted_cluster_namespace> -- env ETCDCTL_API=3
/usr/bin/etcdctl --cacert /etc/etcd/tls/etcd-ca/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
```

4. スナップショットのステータスを確認するには、次のコマンドを実行して、各 etcd コンテナーで **exec** コマンドを使用します。

```
$ oc exec -it <etcd_pod_name> -n <hosted_cluster_namespace> -- env ETCDCTL_API=3
/usr/bin/etcdctl -w table snapshot status /var/lib/data/snapshot.db
```

5. スナップショットデータを、S3 バケットなど、後で取得できる場所にコピーします。以下の例を参照してください。

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。そうしないと、スナップショットを S3 バケットにコピーするときにアップロードが失敗します。

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
```

6. 後で新しいクラスターでスナップショットを復元するには、ホステッド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。

   a. 次のコマンドを実行してシークレットの暗号鍵を取得します。

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

b. 次のコマンドを実行してシークレットの暗号鍵を保存します。

```
$ oc get secret <hosted_cluster_name>-etcd-encryption-key -o=jsonpath='{.data.key}'
```

新しいクラスターでスナップショットを復元するときに、このキーを復号化できます。

次のステップ

etcd スナップショットを復元します。

9.4.2. ホステッドクラスターでの etcd スナップショットの復元

ホステッドクラスターからの 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。

注記

hcp create コマンドの **--render** フラグはシークレットをレンダリングしません。シークレットをレンダリングするには、**hcp create** コマンドで **--render** フラグと **--render-sensitive** フラグの両方を使用する必要があります。

前提条件

ホステッドクラスターで 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。

9.5. AWS でホストされたクラスターの障害復旧

ホステッドクラスターを Amazon Web Services (AWS) 内の同じリージョンに復元できます。たとえば、管理クラスターのアップグレードが失敗し、ホストされたクラスターが読み取り専用状態になっている場合は、障害復旧が必要になります。

重要

Hosted Control Plane 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 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

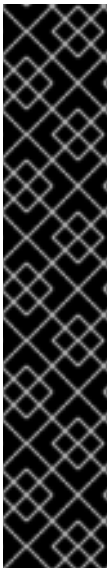
Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

障害復旧プロセスには次の手順が含まれます。

1. ソース管理クラスターでのホステッド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テッドクラスターの復元
3. ホステッド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

API サーバー URL を維持するには、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** フラグが必要です。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。以下の例を参照してください。

例: 外部 DNS フラグ

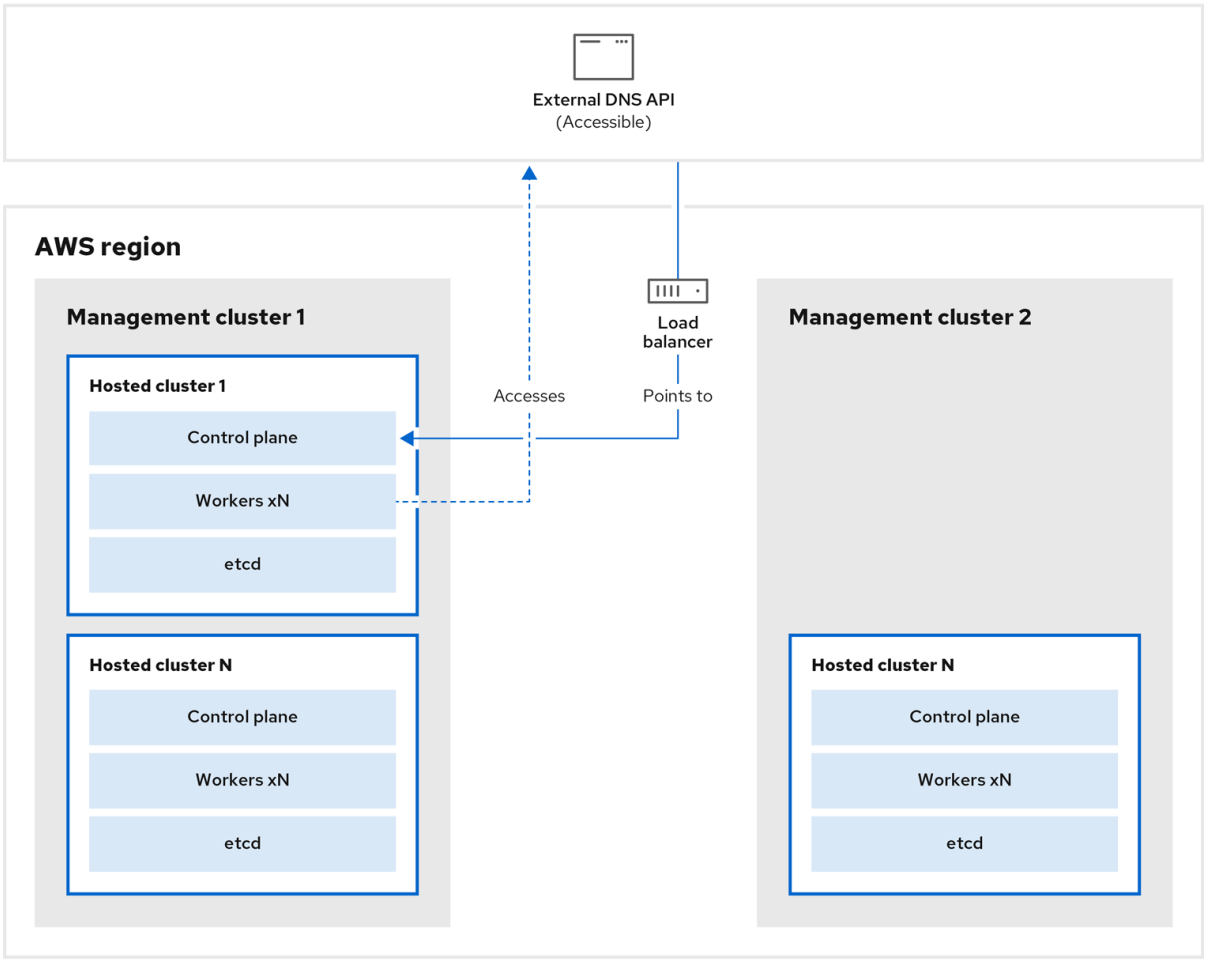
```
--external-dns-provider=aws \
--external-dns-credentials=<path_to_aws_credentials_file> \
--external-dns-domain-filter=<basedomain>
```

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、ホステッドクラスターを移行することはできません。

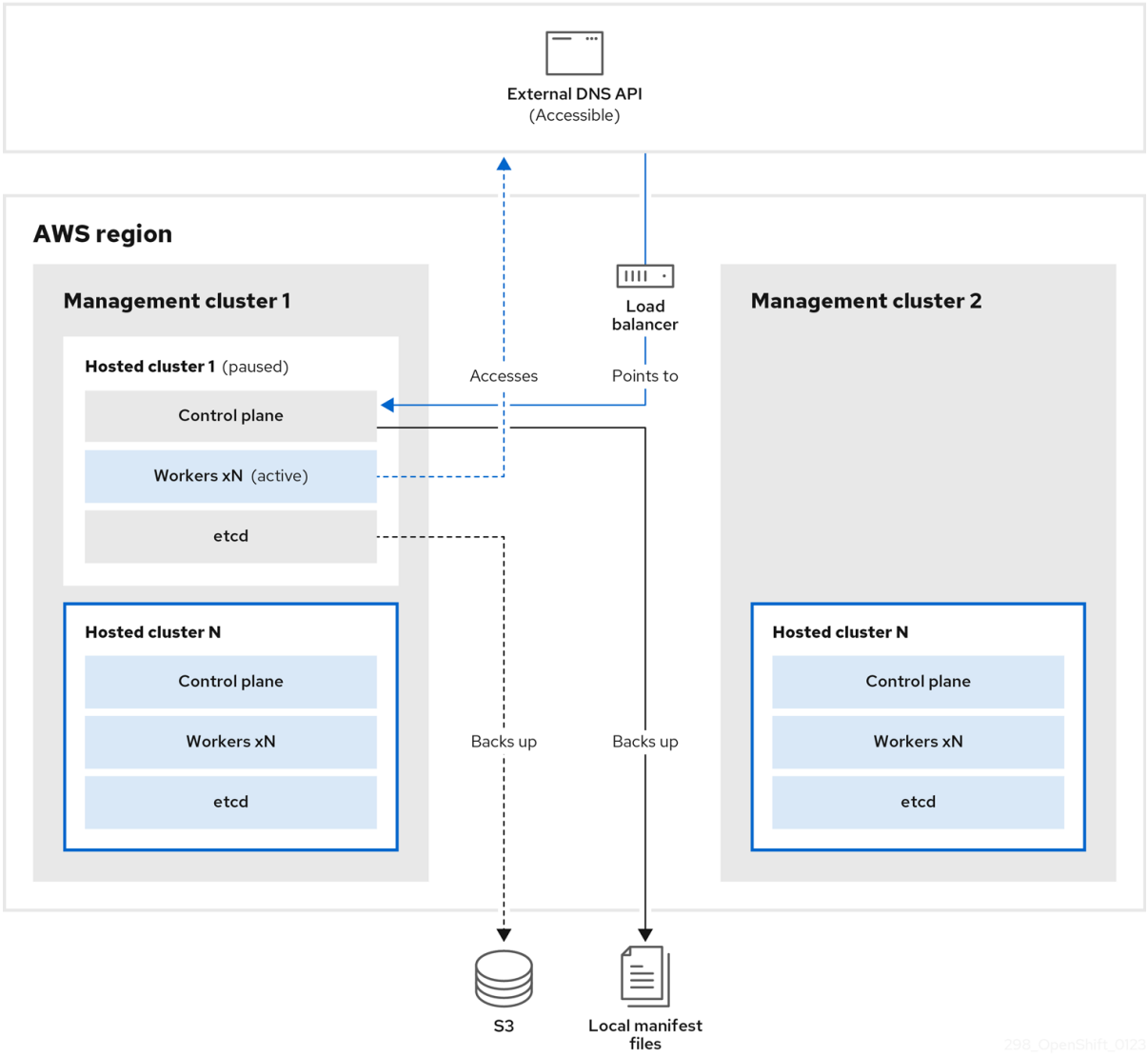
9.5.1. バックアップおよび復元プロセスの概要

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。

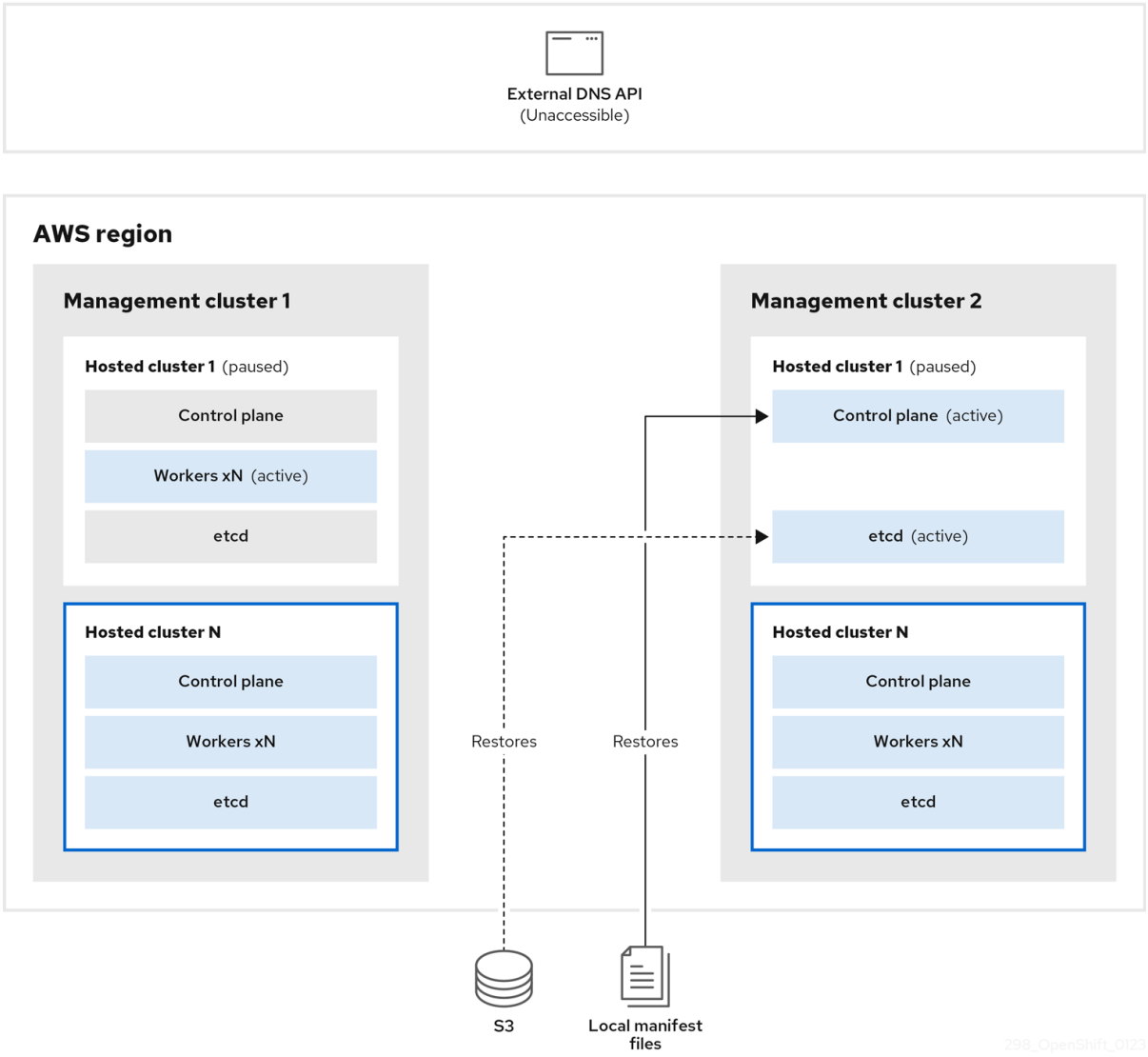
1. 管理クラスター 1 (ソース管理クラスターと見なすことができます) では、コントロールプレーンとワーカーが外部 DNS API を使用して対話します。外部 DNS API はアクセス可能で、管理クラスター間にロードバランサーが配置されています。



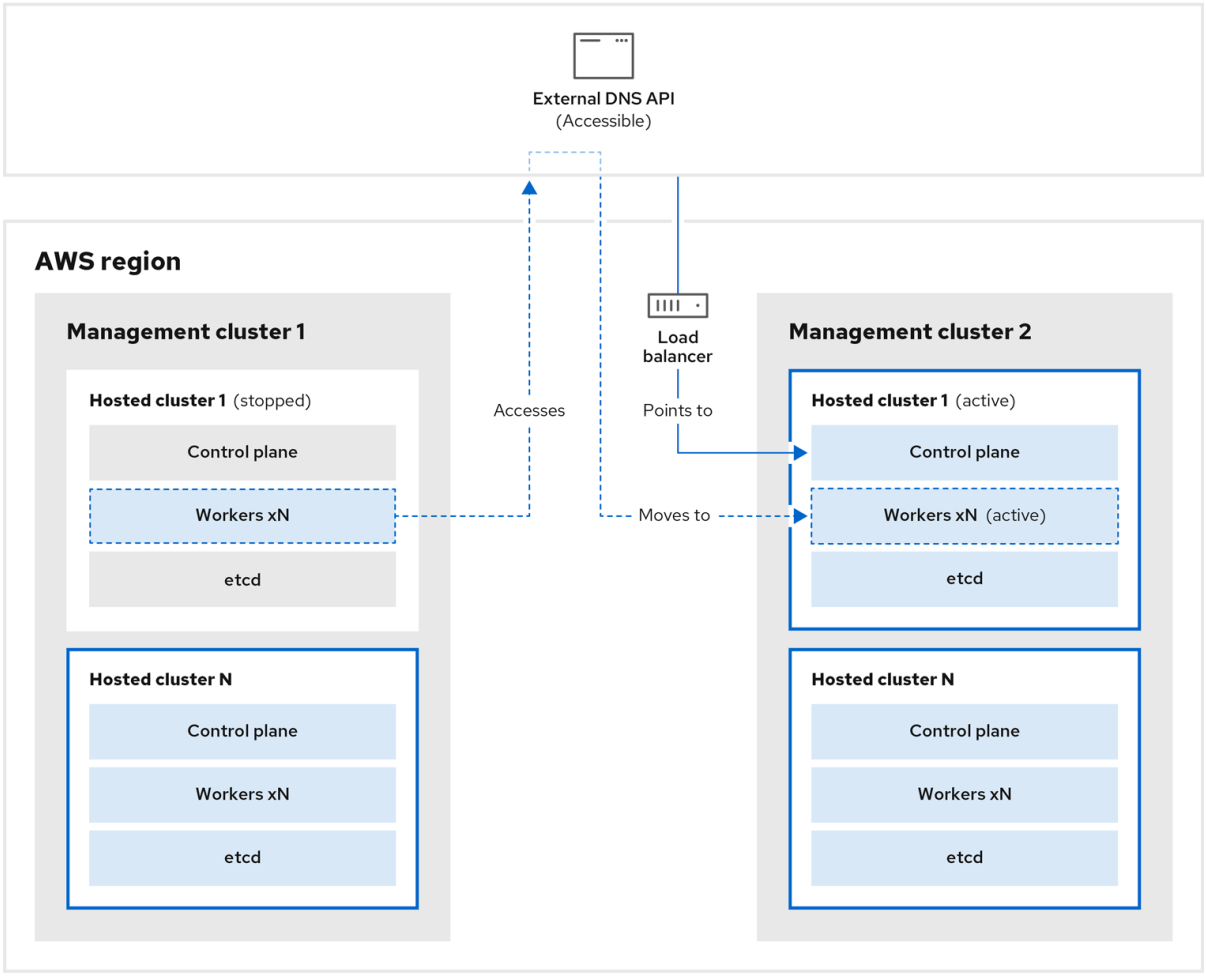
2. ホステッドクラスターのスナップショットを作成します。これには、etcd、コントロールプレーン、およびワーカーノードが含まれます。このプロセスの間、ワーカーノードは外部 DNS API にアクセスできなくても引き続きアクセスを試みます。また、ワークロードが実行され、コントロールプレーンがローカルマニフェストファイルに保存され、etcd が S3 バケットにバックアップされます。データプレーンはアクティブで、コントロールプレーンは一時停止しています。



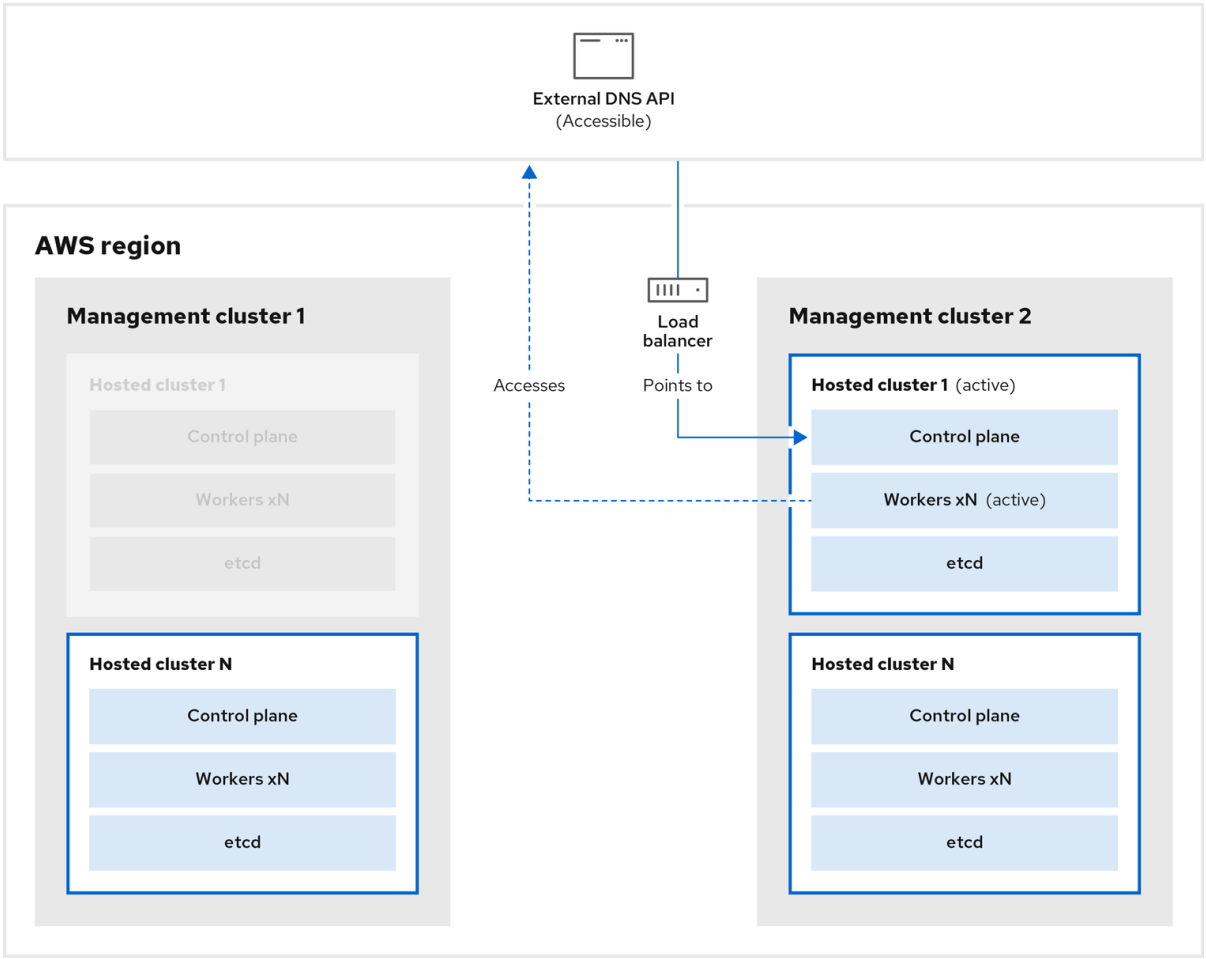
3. 管理クラスター 2 (宛先管理クラスターと見なすことができます) では、S3 バケットから etcd を復元し、ローカルマニフェストファイルからコントロールプレーンを復元します。このプロセスの間、外部 DNS API は停止し、ホステッドクラスター API にアクセスできなくなり、API を使用するワーカーはマニフェストファイルを更新できなくなりますが、ワークロードは引き続き実行されます。



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。ホステッド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。



9.5.2. 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は、「ホステッドクラスターでの 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

ホステッドクラスターを復元します。

9.5.3. ホステッドクラスターの復元

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。

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

```
$ BACKUP_DIR=${HC_CLUSTER_DIR}/backup
```

宛先管理クラスターでの namespace の削除

```
$ 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 作成コマンド

```
$ 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。

シークレットコマンドを復元します。

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

クラスター復元コマンド

```
$ 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 のコマンド

```
$ 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-*
```

マシンのコマンド

```
$ 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 データとホステッド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。

9.5.4. ホステッドクラスターのソース管理クラスターからの削除

ホステッド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テッド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。

重要

spec.persistentVolumeClaimRetentionPolicy.whenScaled フィールドの値が **Delete** に設定されている場合は、データの損失につながる可能性があるため、ステートフルセットをスケーリングしないでください。

回避策として、**spec.persistentVolumeClaimRetentionPolicy.whenScaled** フィールドの値を **Retain** に更新します。ステートフルセットをリコンサイルし、値を **Delete** に戻してしまうようなコントローラーが存在しないことを確認してください。そのようなコントローラーがあると、データの損失が発生する可能性があります。

```
$ export KUBECONFIG=${MGMT_KUBECONFIG}
```

デプロイメントコマンドのスケールダウン

```
$ 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。

```
$ 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 マシンの削除を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。

```
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。

   HCP および ControlPlane HC NS コマンドの削除

   ```
   $ 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。

   HC および HC Namespace コマンドの削除

   ```
   $ 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。

  検証コマンド

  ```
  $ 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}
  ```

  HostedCluster 内でのコマンド

  ```
  $ export KUBECONFIG=${HC_KUBECONFIG}
  ```

  ```
  $ oc get clusterversion
  ```

```
$ oc get nodes
```

次のステップ

ホステッドクラスター内の 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. ホステッドクラスターの 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
```

9.6. OADP を使用したホステッドクラスターの障害復旧

OpenShift API for Data Protection (OADP) Operator を使用して、Amazon Web Services (AWS) およびベアメタルで障害復旧を実行できます。

OpenShift API for Data Protection (OADP) を使用した障害復旧プロセスには、次の手順が含まれます。

1. OADP を使用するために Amazon Web Services やベアメタルなどのプラットフォームを準備
2. データプレーンのワークロードのバックアップ
3. コントロールプレーンのワークロードのバックアップ
4. OADP を使用したホステッドクラスターの復元

9.6.1. 前提条件

管理クラスターで次の前提条件を満たす必要があります。

- OADP Operator をインストールしている。
- ストレージクラスを作成した。
- **cluster-admin** 権限でクラスターにアクセスできる。
- カタログソースを通じて OADP サブスクリプションにアクセスできる。
- S3、Microsoft Azure、Google Cloud、MinIO など、OADP と互換性のあるクラウドストレージプロバイダーにアクセスできる。
- 非接続環境では、OADP と互換性のある Red Hat OpenShift Data Foundation や MinIO などのセルフホスト型ストレージプロバイダーにアクセスできる。
- Hosted Control Plane の Pod が稼働している。

9.6.2. OADP を使用するための AWS の準備

ホステッドクラスターの障害復旧を実行するには、Amazon Web Services (AWS) S3 互換ストレージで OpenShift API for Data Protection (OADP) を使用できます。**DataProtectionApplication** オブジェクトを作成すると、**openshift-adp** namespace に新しい **velero** デプロイメントと **node-agent** Pod が作

成されます。

OADP を使用するために AWS を準備するには、「Multicloud Object Gateway を使用した OpenShift API for Data Protection の設定」を参照してください。

関連情報

- Multicloud Object Gateway を使用した OpenShift API for Data Protection の設定

次のステップ

- データプレーンのワークロードのバックアップ
- コントロールプレーンのワークロードのバックアップ

9.6.3. OADP を使用するためのベアメタルの準備

ホステッドクラスターの障害復旧を実行するには、ベアメタル上で OpenShift API for Data Protection (OADP) を使用できます。**DataProtectionApplication** オブジェクトを作成すると、**openshift-adp** namespace に新しい **velero** デプロイメントと **node-agent** Pod が作成されます。

OADP を使用するためにベアメタルを準備するには、「AWS S3 互換ストレージを使用した OpenShift API for Data Protection の設定」を参照してください。

関連情報

- AWS S3 互換ストレージを使用した OpenShift API for Data Protection の設定

次のステップ

- データプレーンのワークロードのバックアップ
- コントロールプレーンのワークロードのバックアップ

9.6.4. データプレーンのワークロードのバックアップ

データプレーンのワークロードが重要でない場合は、この手順をスキップできます。OADP Operator を使用してデータプレーンワークロードをバックアップするには、「アプリケーションのバックアップ」を参照してください。

関連情報

- アプリケーションのバックアップ

次のステップ

- OADP を使用したホステッドクラスターの復元

9.6.5. コントロールプレーンのワークロードのバックアップ

Backup カスタムリソース (CR) を作成することで、コントロールプレーンのワークロードをバックアップできます。

バックアッププロセスを監視および観察するには、「バックアップおよび復元プロセスの観察」を参照してください。

手順

1. 次のコマンドを実行して、**HostedCluster** リソースの調整を一時停止します。

```
$ oc --kubeconfig <management_cluster_kubeconfig_file> \
  patch hostedcluster -n <hosted_cluster_namespace> <hosted_cluster_name> \
  --type json -p '[{"op": "add", "path": "/spec/pausedUntil", "value": "true"}]'
```

2. 次のコマンドを実行して、**NodePool** リソースの調整を一時停止します。

```
$ oc --kubeconfig <management_cluster_kubeconfig_file> \
  patch nodepool -n <hosted_cluster_namespace> <node_pool_name> \
  --type json -p '[{"op": "add", "path": "/spec/pausedUntil", "value": "true"}]'
```

3. 次のコマンドを実行して、**AgentCluster** リソースのリコンシリエーションを一時停止します。

```
$ oc --kubeconfig <management_cluster_kubeconfig_file> \
  annotate agentcluster -n <hosted_control_plane_namespace>  \
  cluster.x-k8s.io/paused=true --all'
```

4. 次のコマンドを実行して、**AgentMachine** リソースのリコンシリエーションを一時停止します。

```
$ oc --kubeconfig <management_cluster_kubeconfig_file> \
  annotate agentmachine -n <hosted_control_plane_namespace>  \
  cluster.x-k8s.io/paused=true --all'
```

5. 次のコマンドを実行して、**HostedCluster** リソースにアノテーションを付け、Hosted Control Plane namespace が削除されないようにします。

```
$ oc --kubeconfig <management_cluster_kubeconfig_file> \
  annotate hostedcluster -n <hosted_cluster_namespace> <hosted_cluster_name> \
  hypershift.openshift.io/skip-delete-hosted-controlplane-namespace=true
```

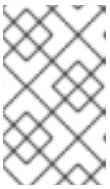
6. **Backup** CR を定義する YAML ファイルを作成します。

例9.1 例: backup-control-plane.yaml ファイル

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: <backup_resource_name> 1
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
spec:
  hooks: {}
  includedNamespaces: 2
  - <hosted_cluster_namespace> 3
  - <hosted_control_plane_namespace> 4
  includedResources:
```

```
    - sa
    - role
    - rolebinding
    - pod
    - pvc
    - pv
    - bmh
    - configmap
    - infraenv 5
    - priorityclasses
    - pdb
    - agents
    - hostedcluster
    - nodepool
    - secrets
    - services
    - deployments
    - hostedcontrolplane
    - cluster
    - agentcluster
    - agentmachinetemplate
    - agentmachine
    - machinedeployment
    - machineset
    - machine
  excludedResources: []
  storageLocation: default
  ttl: 2h0m0s
  snapshotMoveData: true 6
  datamover: "velero" 7
  defaultVolumesToFsBackup: true 8
```

1. **backup_resource_name** を **Backup** リソースの名前に置き換えます。
2. 特定の namespace を選択して、そこからオブジェクトをバックアップします。ホステッドクラスターの namespace と Hosted Control Plane の namespace を含める必要があります。
3. **<hosted_cluster_namespace>** を、ホステッドクラスター namespace の名前 (例: **clusters**) に置き換えます。
4. **<hosted_control_plane_namespace>** は、Hosted Control Plane の namespace の名前 (例: **clusters-hosted**) に置き換えます。
5. **infraenv** リソースを別の namespace に作成する必要があります。バックアッププロセス中に **infraenv** リソースを削除しないでください。
6. 7 CSI ボリュームスナップショットを有効にし、コントロールプレーンのワークロードをクラウドストレージに自動的にアップロードします。
8. 永続ボリューム (PV) の **fs-backup** バックアップ方法をデフォルトとして設定します。この設定は、Container Storage Interface (CSI) ボリュームスナップショットと **fs-backup** 方式を組み合わせて使用する場合に便利です。



注記

CSI ボリュームスナップショットを使用する場合は、PV に **backup.velero.io/backup-volumes-excludes=<pv_name>** アノテーションを追加する必要があります。

7. 次のコマンドを実行して、**Backup** CR を適用します。

```
$ oc apply -f backup-control-plane.yaml
```

検証

- 次のコマンドを実行して、**status.phase** の値が **Completed** になっているかどうかを確認します。

```
$ oc get backups.velero.io <backup_resource_name> -n openshift-adp -o
jsonpath='{.status.phase}'
```

次のステップ

- OADP を使用してホストされたクラスターの復元

9.6.6. OADP を使用してホストされたクラスターの復元

Restore カスタムリソース (CR) を作成することで、ホストされたクラスターを復元できます。

- **インプレース** 更新を使用している場合、InfraEnv にスペアノードは必要ありません。新しい管理クラスターからワーカーノードを再プロビジョニングする必要があります。
- **replace** 更新を使用している場合は、ワーカーノードをデプロイするために InfraEnv 用の予備ノードがいくつか必要です。

重要

ホステッドクラスターをバックアップした後、復元プロセスを開始するには、そのクラスターを破棄する必要があります。ノードのプロビジョニングを開始するには、ホステッドクラスターを削除する前に、データプレーン内のワークロードをバックアップする必要があります。

前提条件

- コンソールを使用したクラスターの削除 の手順に従ってホストされたクラスターを削除している。
- クラスターの削除後に残りのリソースの削除 の手順を完了している。

バックアッププロセスを監視および観察するには、「バックアップおよび復元プロセスの観察」を参照してください。

手順

1. 次のコマンドを実行して、Hosted Control Plane namespace に Pod と永続ボリューム要求 (PVC) が存在しないことを確認します。

```
$ oc get pod pvc -n <hosted_control_plane_namespace>
```

予想される出力

```
No resources found
```

2. **Restore** CR を定義する YAML ファイルを作成します。

restore-hosted-cluster.yaml ファイルの例

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: <restore_resource_name> 1
  namespace: openshift-adp
spec:
  backupName: <backup_resource_name> 2
  restorePVs: true 3
  existingResourcePolicy: update 4
  excludedResources:
  - nodes
  - events
  - events.events.k8s.io
  - backups.velero.io
  - restores.velero.io
  - resticrepositories.velero.io
```

1. **<restore_resource_name>** を **Restore** リソースの名前に置き換えます。
2. **<backup_resource_name>** を **Backup** リソースの名前に置き換えます。
3. 永続ボリューム (PV) とその Pod のリカバリーを開始します。
4. 既存のオブジェクトがバックアップされたコンテンツで上書きされるようにします。

重要

infraenv リソースを別の namespace に作成する必要があります。復元プロセス中に **infraenv** リソースを削除しないでください。新しいノードを再プロビジョニングするには、**infraenv** リソースが必須です。

3. 次のコマンドを実行して、**Restore** CR を適用します。

```
$ oc apply -f restore-hosted-cluster.yaml
```

4. 次のコマンドを実行して、**status.phase** の値が **Completed** になっているかどうかを確認します。

```
$ oc get hostedcluster <hosted_cluster_name> -n <hosted_cluster_namespace> -o
jsonpath='{.status.phase}'
```

5. 復元プロセスが完了したら、コントロールプレーンワークロードのバックアップ中に一時停止した **HostedCluster** リソースおよび **NodePool** リソースのリコンシリエーションを開始します。

   a. 次のコマンドを実行して、**HostedCluster** リソースのリコンシリエーションを開始します。

   ```
   $ oc --kubeconfig <management_cluster_kubeconfig_file> \
     patch hostedcluster -n <hosted_cluster_namespace> <hosted_cluster_name> \
     --type json -p '[{"op": "add", "path": "/spec/pausedUntil", "value": "false"}]'
   ```

   b. 次のコマンドを実行して、**NodePool** リソースのリコンシリエーションを開始します。

   ```
   $ oc --kubeconfig <management_cluster_kubeconfig_file> \
     patch nodepool -n <hosted_cluster_namespace> <node_pool_name> \
     --type json -p '[{"op": "add", "path": "/spec/pausedUntil", "value": "false"}]'
   ```

6. コントロールプレーンワークロードのバックアップ中に一時停止したエージェントプロバイダーリソースのリコンシリエーションを開始します。

   a. 次のコマンドを実行して、**AgentCluster** リソースのリコンシリエーションを開始します。

   ```
   $ oc --kubeconfig <management_cluster_kubeconfig_file> \
     annotate agentcluster -n <hosted_control_plane_namespace>  \
     cluster.x-k8s.io/paused- --overwrite=true --all
   ```

   b. 次のコマンドを実行して、**AgentMachine** リソースのリコンシリエーションを開始します。

   ```
   $ oc --kubeconfig <management_cluster_kubeconfig_file> \
     annotate agentmachine -n <hosted_control_plane_namespace>  \
     cluster.x-k8s.io/paused- --overwrite=true --all
   ```

7. 次のコマンドを実行して、**HostedCluster** リソースの **hypershift.openshift.io/skip-delete-hosted-controlplane-namespace-** アノテーションを削除し、Hosted Control Plane namespace の手動削除を回避します。

   ```
   $ oc --kubeconfig <management_cluster_kubeconfig_file> \
     annotate hostedcluster -n <hosted_cluster_namespace> <hosted_cluster_name> \
     hypershift.openshift.io/skip-delete-hosted-controlplane-namespace- \
     --overwrite=true --all
   ```

8. 次のコマンドを実行して、**NodePool** リソースを必要なレプリカ数にスケーリングします。

   ```
   $ oc --kubeconfig <management_cluster_kubeconfig_file> \
     scale nodepool -n <hosted_cluster_namespace> <node_pool_name> \
     --replicas <replica_count> 1
   ```

   1 **<replica_count>** を整数値 (例: **3**) に置き換えます。

9.6.7. バックアップと復元のプロセスの観察

OpenShift API for Data Protection (OADP) を使用してホステッドクラスターをバックアップおよび復元する場合は、プロセスを監視および観察できます。

手順

1. 次のコマンドを実行して、バックアッププロセスを確認します。

```
$ watch "oc get backups.velero.io -n openshift-adp <backup_resource_name> -o jsonpath='{.status}'"
```

2. 次のコマンドを実行して、復元プロセスを確認します。

```
$ watch "oc get restores.velero.io -n openshift-adp <backup_resource_name> -o jsonpath='{.status}'"
```

3. 次のコマンドを実行して、Velero ログを確認します。

```
$ oc logs -n openshift-adp -ldeploy=velero -f
```

4. 次のコマンドを実行して、すべての OADP オブジェクトの進行状況を確認します。

```
$ watch "echo BackupRepositories:;echo;oc get backuprepositories.velero.io -A;echo; echo BackupStorageLocations: ;echo; oc get backupstoragelocations.velero.io -A;echo;echo DataUploads: ;echo;oc get datauploads.velero.io -A;echo;echo DataDownloads: ;echo;oc get datadownloads.velero.io -n openshift-adp; echo;echo VolumeSnapshotLocations: ;echo;oc get volumesnapshotlocations.velero.io -A;echo;echo Backups:;echo;oc get backup -A; echo;echo Restores:;echo;oc get restore -A"
```

9.6.8. velero CLI を使用してバックアップおよび復元リソースを記述する

OpenShift API for Data Protection を使用する場合は、**velero** コマンドラインインターフェイス (CLI) を使用して、**Backup** および **Restore** リソースの詳細を取得できます。

手順

1. 次のコマンドを実行して、コンテナーから **velero** CLI を使用するためのエイリアスを作成します。

```
$ alias velero='oc -n openshift-adp exec deployment/velero -c velero -it -- ./velero'
```

2. 次のコマンドを実行して、**Restore** カスタムリソース (CR) の詳細を取得します。

```
$ velero restore describe <restore_resource_name> --details 1
```

1 **<restore_resource_name>** を **Restore** リソースの名前に置き換えます。

3. 次のコマンドを実行して、**Backup** CR の詳細を取得します。

```
$ velero restore describe <backup_resource_name> --details 1
```

1 **<backup_resource_name>** を **Backup** リソースの名前に置き換えます。

第10章 HOSTED CONTROL PLANE のトラブルシューティング

Hosted Control Plane で問題が発生した場合は、次の情報を参照してトラブルシューティングを行ってください。

10.1. HOSTED CONTROL PLANE のトラブルシューティング用の情報収集

Hosted Control Plane クラスターの問題をトラブルシューティングする必要がある場合は、**must-gather** コマンドを実行して情報を収集できます。このコマンドは、管理クラスターとホステッドクラスターの出力を生成します。

管理クラスターの出力には次の内容が含まれます。

- **クラススコープのリソース**: これらのリソースは、管理クラスターのノード定義です。
- **hypershift-dump 圧縮ファイル**: このファイルは、コンテンツを他の人と共有する必要がある場合に役立ちます。
- **namespace リソース**: これらのリソースには、config map、サービス、イベント、ログなど、関連する namespace のすべてのオブジェクトが含まれます。
- **ネットワークログ**: これらのログには、OVN ノースバウンドデータベースとサウスバウンドデータベース、およびそれぞれのステータスが含まれます。
- **ホステッドクラスター**: このレベルの出力には、ホステッドクラスター内のすべてのリソースが含まれます。

ホステッドクラスターの出力には、次の内容が含まれます。

- **クラススコープのリソース**: これらのリソースには、ノードや CRD などのクラスター全体のオブジェクトがすべて含まれます。
- **namespace リソース**: これらのリソースには、config map、サービス、イベント、ログなど、関連する namespace のすべてのオブジェクトが含まれます。

出力にはクラスターからのシークレットオブジェクトは含まれませんが、シークレットの名前への参照が含まれる可能性があります。

前提条件

- 管理クラスターへの **cluster-admin** アクセス権がある。
- **HostedCluster** リソースの **name** 値と、CR がデプロイされる namespace がある。
- **hcp** コマンドラインインターフェイスがインストールされている。詳細は、Hosted Control Plane のコマンドラインインターフェイスをインストールする を参照してください。
- OpenShift CLI (**oc**) がインストールされている。
- **kubeconfig** ファイルがロードされ、管理クラスターを指している。

手順

- トラブルシューティングのために出力を収集するには、次のコマンドを入力します。

```
$ oc adm must-gather --image=registry.redhat.io/multicluster-engine/must-gather-rhel9:v<mce_version> \
  /usr/bin/gather hosted-cluster-namespace=HOSTEDCLUSTERNAMESPACE hosted-cluster-name=HOSTEDCLUSTERNAME \
  --dest-dir=NAME ; tar -cvzf NAME.tgz NAME
```

ここでは、以下のようになります。

- **<mce_version>** は、使用している multicluster engine Operator のバージョン (例: **2.6**) に置き換えます。
- **hosted-cluster-namespace=HOSTEDCLUSTERNAMESPACE** パラメーターはオプションです。このパラメーターを使用しないと、ホストされたクラスターがデフォルトの namespace (**clusters**) にあるかのようにコマンドが実行します。
- **--dest-dir=NAME** パラメーターはオプションです。コマンドの結果を圧縮ファイルに保存する場合は、そのパラメーターを指定して、**NAME** を、結果を保存するディレクトリーの名前に置き換えます。

10.2. HOSTED CONTROL PLANE コンポーネントの再起動

Hosted Control Plane の管理者の場合は、**hypershift.openshift.io/restart-date** アノテーションを使用して、特定の **HostedCluster** リソースのすべてのコントロールプレーンコンポーネントを再起動できます。たとえば、証明書のローテーション用にコントロールプレーンコンポーネントを再起動する必要がある場合があります。

手順

- コントロールプレーンを再起動するには、次のコマンドを入力して **HostedCluster** リソースにアノテーションを付けます。

```
$ oc annotate hostedcluster \
  -n <hosted_cluster_namespace> \
  <hosted_cluster_name> \
  hypershift.openshift.io/restart-date=$(date --iso-8601=seconds) 1
```

1 アノテーションの値が変わるたびに、コントロールプレーンが再起動されます。**date** コマンドは、一意の文字列のソースとして機能します。アノテーションはタイムスタンプではなく文字列として扱われます。

検証

コントロールプレーンを再起動すると、通常、次の Hosted Control Plane コンポーネントが再起動します。

注記

他のコンポーネントによって変更が実装されることに伴い、さらにいくつかのコンポーネントが再起動する場合があります。

- catalog-operator
- certified-operators-catalog

- cluster-api
- cluster-autoscaler
- cluster-policy-controller
- cluster-version-operator
- community-operators-catalog
- control-plane-operator
- hosted-cluster-config-operator
- ignition-server
- ingress-operator
- konnectivity-agent
- konnectivity-server
- kube-apiserver
- kube-controller-manager
- kube-scheduler
- machine-approver
- oauth-openshift
- olm-operator
- openshift-apiserver
- openshift-controller-manager
- openshift-oauth-apiserver
- packageserver
- redhat-marketplace-catalog
- redhat-operators-catalog

10.3. ホステッドクラスターと HOSTED CONTROL PLANE のリコンシリエーションの一時停止

クラスターインスタンス管理者は、ホステッドクラスターと Hosted Control Plane のリコンシリエーションを一時停止できます。etcd データベースをバックアップおよび復元するときや、ホステッドクラスターまたは Hosted Control Plane の問題をデバッグする必要があるときは、リコンシリエーションを一時停止することができます。

手順

1. ホステッドクラスターと Hosted Control Plane のリコンシリエーションを一時停止するには、**HostedCluster** リソースの **pausedUntil** フィールドを設定します。

   - 特定の時刻までリコンシリエーションを一時停止するには、次のコマンドを入力します。

     ```
     $ oc patch -n <hosted_cluster_namespace> hostedclusters/<hosted_cluster_name> -p '{"spec":{"pausedUntil":"<timestamp>"}}' --type=merge 1
     ```

     1 RFC339 形式でタイムスタンプを指定します (例: **2024-03-03T03:28:48Z**)。指定の時間が経過するまで、リコンシリエーションが一時停止します。

   - リコンシリエーションを無期限に一時停止するには、次のコマンドを入力します。

     ```
     $ oc patch -n <hosted_cluster_namespace> hostedclusters/<hosted_cluster_name> -p '{"spec":{"pausedUntil":"true"}}' --type=merge
     ```

     HostedCluster リソースからフィールドを削除するまで、リコンシリエーションは一時停止されます。

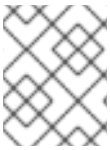
     HostedCluster リソースの一時停止リコンシリエーションフィールドが設定されると、そのフィールドは関連付けられた **HostedControlPlane** リソースに自動的に追加されます。

2. **pausedUntil** フィールドを削除するには、次の patch コマンドを入力します。

   ```
   $ oc patch -n <hosted_cluster_namespace> hostedclusters/<hosted_cluster_name> -p '{"spec":{"pausedUntil":null}}' --type=merge
   ```

10.4. データプレーンをゼロにスケールダウンする

Hosted Control Plane を使用していない場合は、リソースとコストを節約するために、データプレーンをゼロにスケールダウンできます。



注記

データプレーンをゼロにスケールダウンする準備ができていることを確認してください。スケールダウンするとワーカーノードからのワークロードがなくなるためです。

手順

1. 次のコマンドを実行して、ホステッドクラスターにアクセスするように **kubeconfig** ファイルを設定します。

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

2. 次のコマンドを実行して、ホステッドクラスターに関連付けられた **NodePool** リソースの名前を取得します。

   ```
   $ oc get nodepool --namespace <HOSTED_CLUSTER_NAMESPACE>
   ```

3. オプション: Pod のドレインを防止するには、次のコマンドを実行して、**NodePool** リソースに **nodeDrainTimeout** フィールドを追加します。

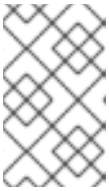
```
$ oc edit nodepool <nodepool_name>  --namespace <hosted_cluster_namespace>
```

出力例

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
# ...
  name: nodepool-1
  namespace: clusters
# ...
spec:
  arch: amd64
  clusterName: clustername (1)
  management:
    autoRepair: false
    replace:
      rollingUpdate:
        maxSurge: 1
        maxUnavailable: 0
      strategy: RollingUpdate
    upgradeType: Replace
  nodeDrainTimeout: 0s (2)
# ...
```

(1) ホステッドクラスターの名前を定義します。

(2) コントローラーがノードをドレインするのに費やす合計時間を指定します。デフォルトでは、**nodeDrainTimeout: 0s** 設定はノードドレインプロセスをブロックします。



注記

ノードドレインプロセスを一定期間継続できるようにするには、それに応じて、**nodeDrainTimeout** フィールドの値を設定できます (例: **nodeDrainTimeout: 1m**)。

4. 次のコマンドを実行して、ホステッドクラスターに関連付けられた **NodePool** リソースをスケールダウンします。

```
$ oc scale nodepool/<NODEPOOL_NAME> --namespace
<HOSTED_CLUSTER_NAMESPACE> --replicas=0
```

注記

データプランをゼロにスケールダウンした後、コントロールプレーン内の一部の Pod は **Pending** ステータスのままになり、ホストされているコントロールプレーンは稼働したままになります。必要に応じて、**NodePool** リソースをスケールアップできます。

5. オプション: 次のコマンドを実行して、ホステッドクラスターに関連付けられた **NodePool** リソースをスケールアップします。

```
$ oc scale nodepool/<NODEPOOL_NAME> --namespace
<HOSTED_CLUSTER_NAMESPACE> --replicas=1
```

NodePool リソースを再スケーリングした後、**NodePool** リソースが準備 **Ready** 状態で使用可能になるまで数分間待ちます。

検証

- 次のコマンドを実行して、**nodeDrainTimeout** フィールドの値が **0s** より大きいことを確認します。

```
$ oc get nodepool -n <hosted_cluster_namespace> <nodepool_name> -
ojsonpath='{.spec.nodeDrainTimeout}'
```

関連情報

- ホステッドクラスターの must-gather