OpenShift Container Platform 4.16

Hosted Control Plane

OpenShift Container Platform で Hosted Control Plane を使用する

Last Updated: 2024-07-02

OpenShift Container Platform 4.16 Hosted Control Plane

OpenShift Container Platform で Hosted Control Plane を使用する

法律上の通知

概要

このドキュメントでは、OpenShift Container Platform の 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。

目次

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構成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

ホストされたクラスター

管理クラスター上でホストされるコントロールプレーンと API エンドポイントを備えた OpenShift Container Platform クラスター。ホストされたクラスターには、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。

ホストされたクラスター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、コンピューティング、およびストレージリソース。

Hosted Control Plane

管理クラスター上で実行している OpenShift Container Platform コントロールプレーン。ホストされたクラスターの 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、Kubernetes Operator のマルチクラスターエンジンが Red Hat Advanced Cluster Management で管理するクラスターのライフサイクルに固有のものです。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、ホストされたクラスターのコントロールプレーンがホストされる OpenShift Container Platform クラスター。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、コンピューティング、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、ホストされた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、論理境界内でホストされた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 とテナントのホストされた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)

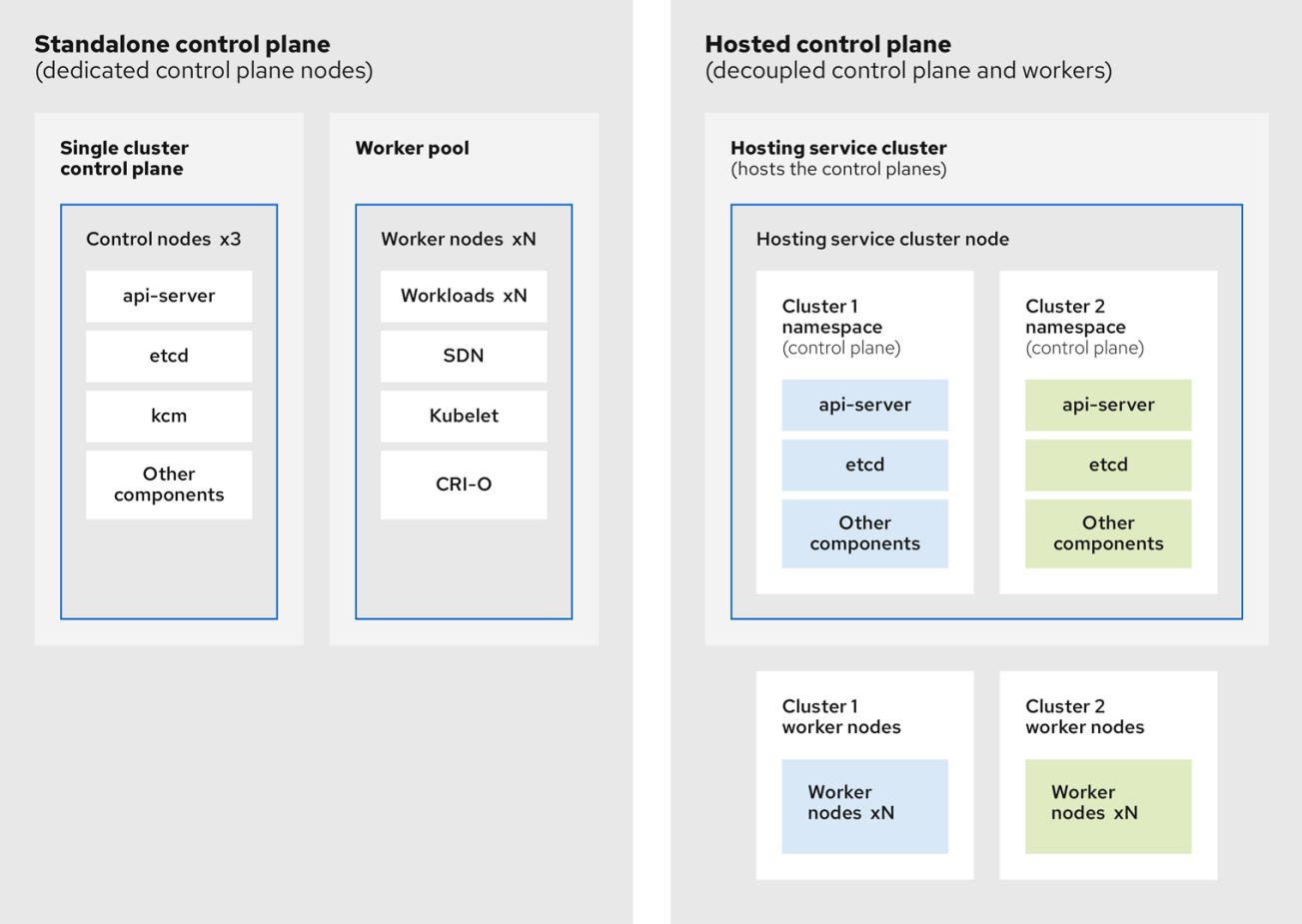
1.2.1. Hosted control plane のアーキテクチャー

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設定される結合モデルまたはスタンドアロンモデルでデプロイされます。コントロールプレーンには、API エンドポイント、ストレージエンドポイント、ワークロードスケジューラー、および状態を保

証するアクチュエーターが含まれます。データプレーンには、ワークロードとアプリケーションが実行されるコンピューティング、ストレージ、ネットワークが含まれます。

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

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



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 で設定されるため、コントロールプレーンはすぐに起動します。同じ原則が、モニタリング、ロギング、自動スケーリングなどのコントロールプレーンとワークロードに適用されます。
- インフラストラクチャーの観点からは、レジストリー、HAProxy、クラスター監視、ストレージノードなどのインフラストラクチャーをテナントのクラウドプロバイダーのアカウントにプッシュして、テナントでの使用を分離できます。
- 運用上の観点からは、マルチクラスター管理はさらに集約され、クラスターの状態と一貫性に影響を与える外部要因が少なくなります。Site Reliability Engineer は、一箇所で問題をデバッグして、クラスターのデータプレインを移動するため、解決までの時間 (TTR) が短縮され、生産性が向上します。

関連情報

- Hosted Control Plane

1.3. 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 オペレーターは、**hypershift** namespace に **supported-versions** config map を作成します。config map には、サポートされているホスト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、ホストされた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。
- ベアメタル上で Hosted Control Plane クラスターを設定する
  - 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)

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

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

- 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 Hosted Control Plane クラスターを設定する (テクノロジープレビュー) : AWS で Hosted Control Plane 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。
- ホステッドコントロールプレーン用の 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

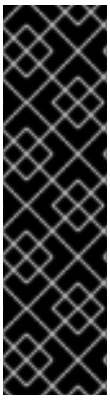
IBM Z プラットフォーム上の Hosted Control Plane は、テクノロジープレビュー機能としてのみ利用できます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

- 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



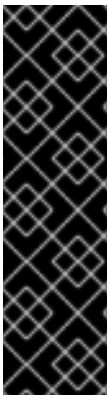
重要

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

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

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

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

- 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。ホストされた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。

前提条件

- ホストされた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 <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. ホストされたクラスターの名前を指定します。
2. ホストされたクラスターの 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トされたクラスターの 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。

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。
- ホストされた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. ホストされた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

- サポート対象のアイデンティティープロバイダーに関する詳細は、**認証および承認** 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。

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。

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。

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

4.2. 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

ホストされたクラスター内のノードでノードレベルのチューニングを設定するには、Node Tuning Operator を使用できます。ホストされたコントロールプレーンでは、**Tuned** オブジェクトを含む 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。ホストされたコントロールプレーンでは、Node Tuning Operator はノードプール名と namespace のハッシュを Tuned CR の名前に追加してそれらを区別します。このケース以外では、同じホストクラスターの異なる Tuned CR に同じ名前の複数の TuneD プロファイルを作成しないでください。

検証

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

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

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

出力例

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

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

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

出力例

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

注記

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

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

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

出力例

```
vm.dirty_ratio = 55
```

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

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

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: s/qe-app-registry/redhat-operators
  sourceNamespace: openshift-marketplace
```

検証

1. SR-IOV Operator の準備ができていることを確認するには、次のコマンドを実行し、結果の出力を表示します。

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

出力例

```
NAME                                         DISPLAY                   VERSION             REPLACES
PHASE
sriov-network-operator.4.16.0-202211021237   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
```

第5章 ホストされたクラスターでのフィーチャーゲートの使用

ホストされた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。ホストされた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、ホストされたクラスターで **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 <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. ホストされたクラスターの名前を指定します。
2. ホストされたクラスターの 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、ホストされたクラスターで **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 の更新

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

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

Kubernetes Operator のマルチクラスターエンジンは、1 つ以上の OpenShift Container Platform クラスターを管理できます。OpenShift Container Platform でホストされたクラスターを作成した後、ホストされたクラスターをマネージドクラスターとしてマルチクラスターエンジン 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。
- ホストされた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。ホストされた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. マルチクラスターエンジン 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. ホストされた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、「Hosted Control Plane のノードプールの更新」および「Hosted Control Plane 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 Operator の更新

6.2. ホストされたクラスターの更新

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

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。

6.3. ノードプールの更新

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

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

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

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

6.3.1. ノードプールの置き換え更新

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

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

6.3.2. 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。

6.4. HOSTED CONTROL PLANE のノードプールの更新

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

手順

- 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トされたクラスターの 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-**

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。

6.5. HOSTED CONTROL PLANE のホストクラスターの更新

Hosted Control Plane では、ホストされた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。

手順

1. 次のコマンドを実行して、ホストされたクラスターに **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、それぞれホストされたクラスター名とホストされたクラスター 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、ホストされたクラスターの **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

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、ホストされたクラスターの **.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。

第7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、管理対象のホストされたクラスターごとに、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。

7.1. 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、メトリックの最小セットです。
- **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
```

7.1.1. SRE メトリックセットの設定

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ごとに編成された **RelabelConfig** 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__"]
```

7.2. ホストされたクラスターのモニタリングダッシュボードの有効化

ホストされたクラスターのモニタリングダッシュボードを有効にするには、次の手順を実行します。

手順

手順

1. **local-cluster** namespace に **hypershift-operator-install-flags** config map を作成します。**data.installFlagsToAdd** セクションで **--monitoring-dashboards** フラグを必ず指定してください。以下に例を示します。

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

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 が管理するホストされたクラスターごとに、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>** はホストされたクラスターの namespace、**<hosted_cluster_name>** はホストされた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、ホストされた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: ホストされた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** フラグを削除します。ホストされた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。

7.2.1. ダッシュボードのカスタマイズ

各ホストされた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トされた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__	ホストされたクラスターの名前
__NAMESPACE__	ホストされたクラスターの名前空間
__CONTROL_PLANE_NAMESPACE__	ホストされた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__	ホストされたクラスターの UUID。ホストされたクラスターのメトリクスの **_id** ラベルと一致します。

第8章 HOSTED CONTROL PLANE の高可用性

8.1. 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。

8.1.1. 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 <hosted_control_plane_namespace> -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 health --cluster -w table
```

出力例

```
ENDPOINT                                        HEALTH  TOOK       ERROR
https://etcd-0.etcd-discovery.clusters-hosted.svc:2379  true    9.117698ms
```

8.1.2. 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 <hosted_control_plane_namespace>
```

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 pvc/<etcd_pvc_name> pod/<etcd_pod_name> --wait=false
```

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 <hosted_control_plane_namespace>
```

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
```

8.2. 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トされたクラスターで etcd をバックアップおよび復元して、障害を修正できます。

8.2.1. オンプレミス環境のホストされたクラスターでの etcd のバックアップと復元

ホストされたクラスターで etcd をバックアップおよび復元することで、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 ${CLUSTER_NAMESPACE} hostedclusters/${CLUSTER_NAME} -p '{"spec":
{"pausedUntil":""}}' --type=merge
```

8.3. 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。

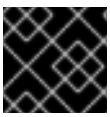
重要

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

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

8.3.1. 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

ホストされたクラスターの 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、ホストされたクラスターの調整を一時停止します。

```
$ oc patch -n clusters hostedclusters/<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
```

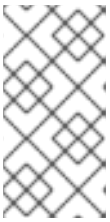
3. etcd スナップショットを取得するには、次のコマンドを実行して、各 etcd コンテナーで **exec** コマンドを使用します。

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

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、ホストされたクラスターが参照する暗号化シークレットを保存します。

   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。

8.3.2. ホストされたクラスターでの etcd スナップショットの復元

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

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

前提条件

ホストされたクラスターで etcd スナップショットを作成している。

手順

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

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

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

   ```
   spec:
     etcd:
       managed:
         storage:
           persistentVolume:
             size: 4Gi
           type: PersistentVolume
   ```

```
    restoreSnapshotURL:
    - "${ETCD_SNAPSHOT_URL}"
managementType: Managed
```

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

8.4. AWS でホストされたクラスターの障害復旧

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

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

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トされたクラスターのバックアップ
2. 宛先管理クラスターでのホストされたクラスターの復元
3. ホストされたクラスターのソース管理クラスターからの削除

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

重要

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

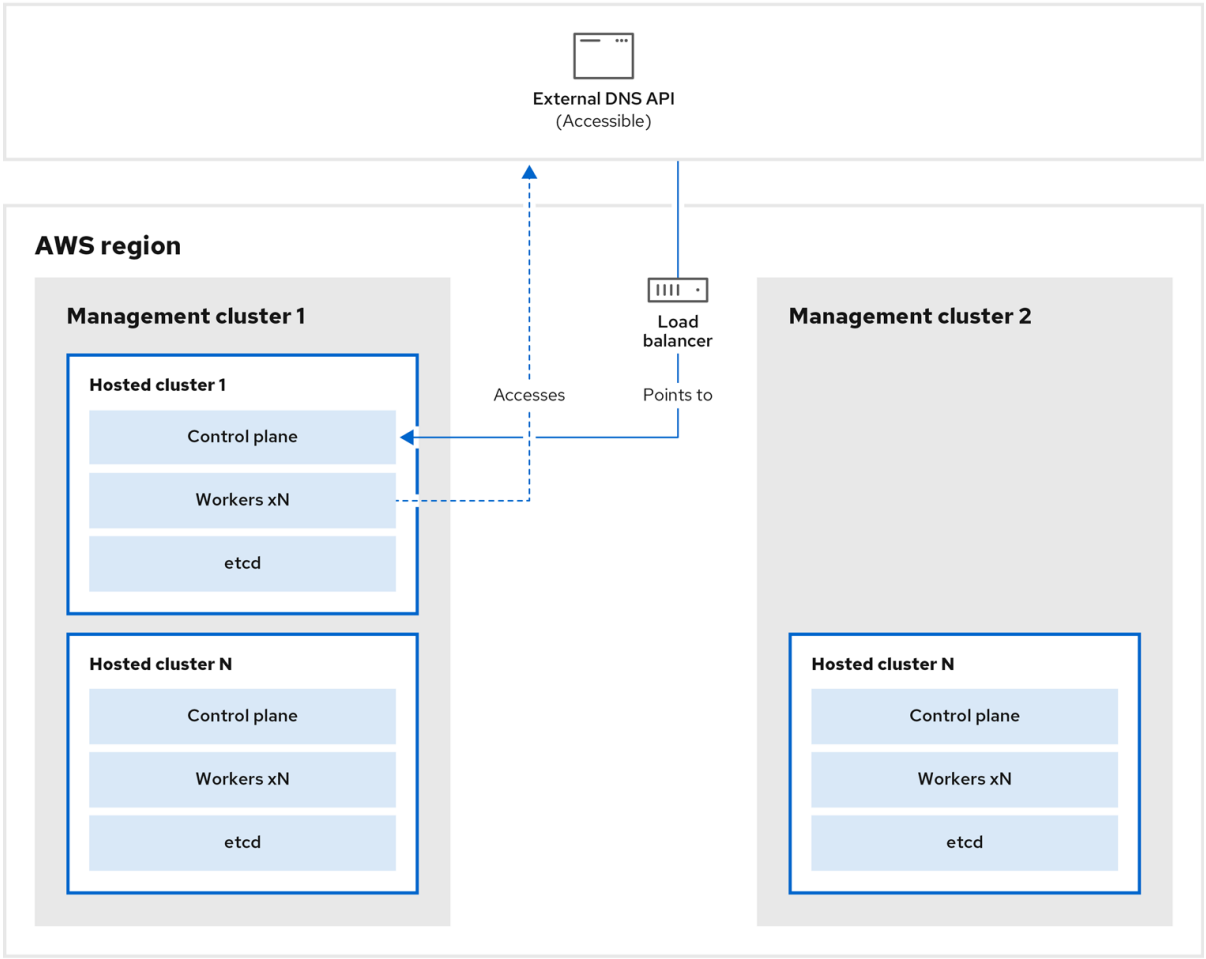
```
--external-dns-provider=aws \
--external-dns-credentials=<path_to_aws_credentials_file> \
--external-dns-domain-filter=<basedomain>
```

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、ホストされたクラスターを移行することはできません。

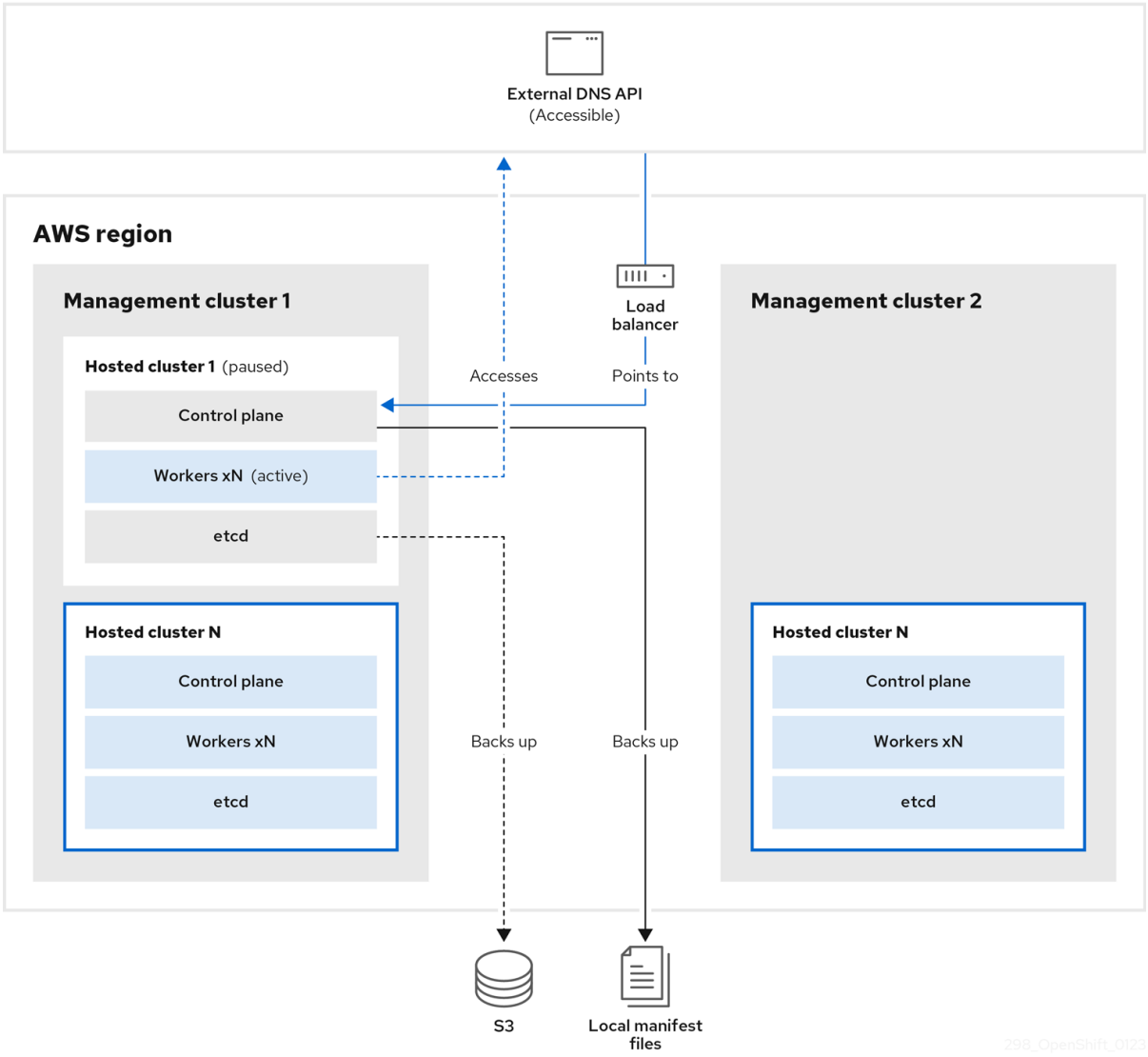
8.4.1. バックアップおよび復元プロセスの概要

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。

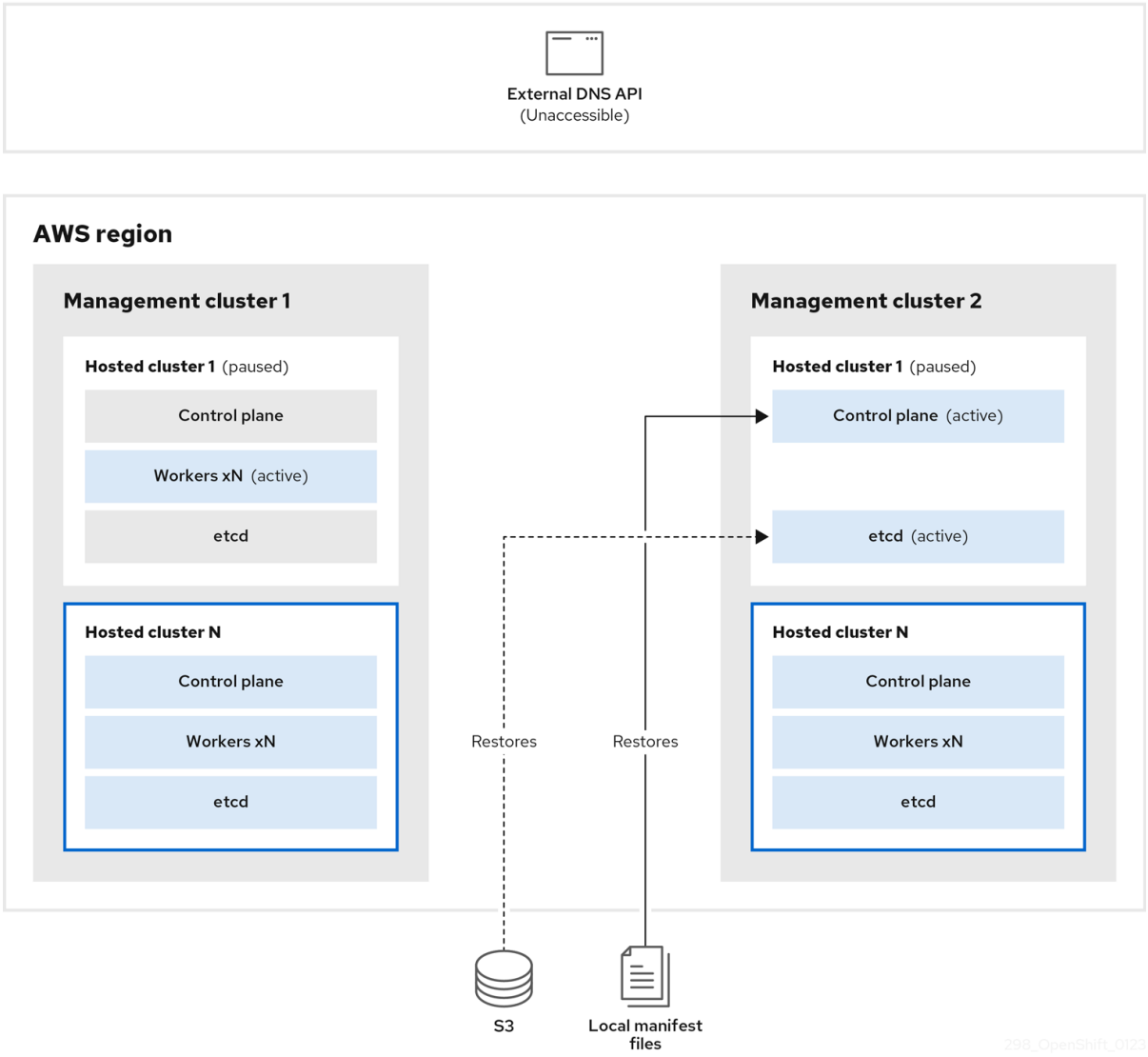
1. 管理クラスター 1 (ソース管理クラスターと見なすことができます) では、コントロールプレーンとワーカーが外部 DNS API を使用して対話します。外部 DNS API はアクセス可能で、管理クラスター間にロードバランサーが配置されています。



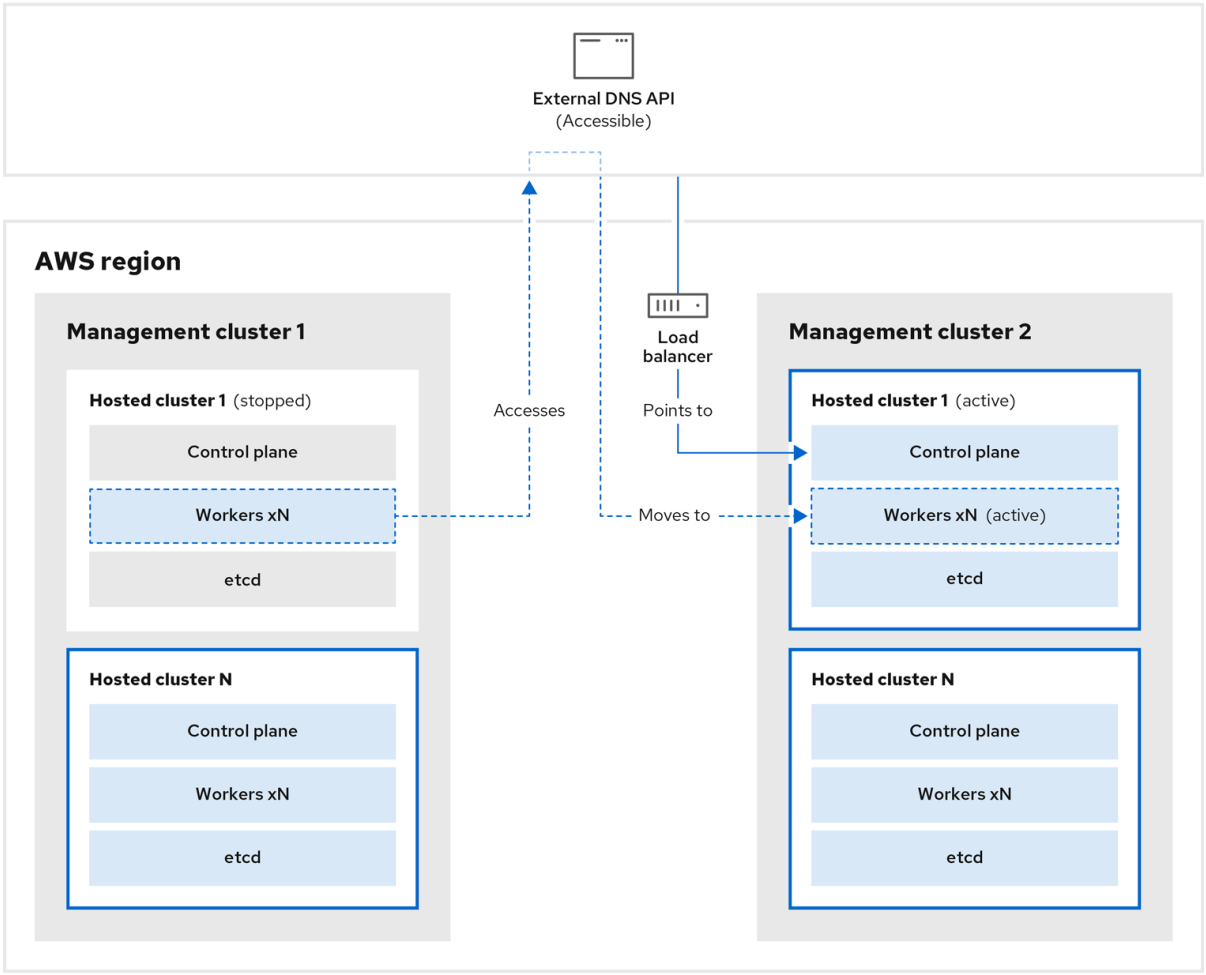
2. ホストされたクラスターのスナップショットを作成します。これには、etcd、コントロールプレーン、およびワーカーノードが含まれます。このプロセスの間、ワーカーノードは外部 DNS API にアクセスできなくても引き続きアクセスを試みます。また、ワークロードが実行され、コントロールプレーンがローカルマニフェストファイルに保存され、etcd が S3 バケットにバックアップされます。データプレーンはアクティブで、コントロールプレーンは一時停止しています。



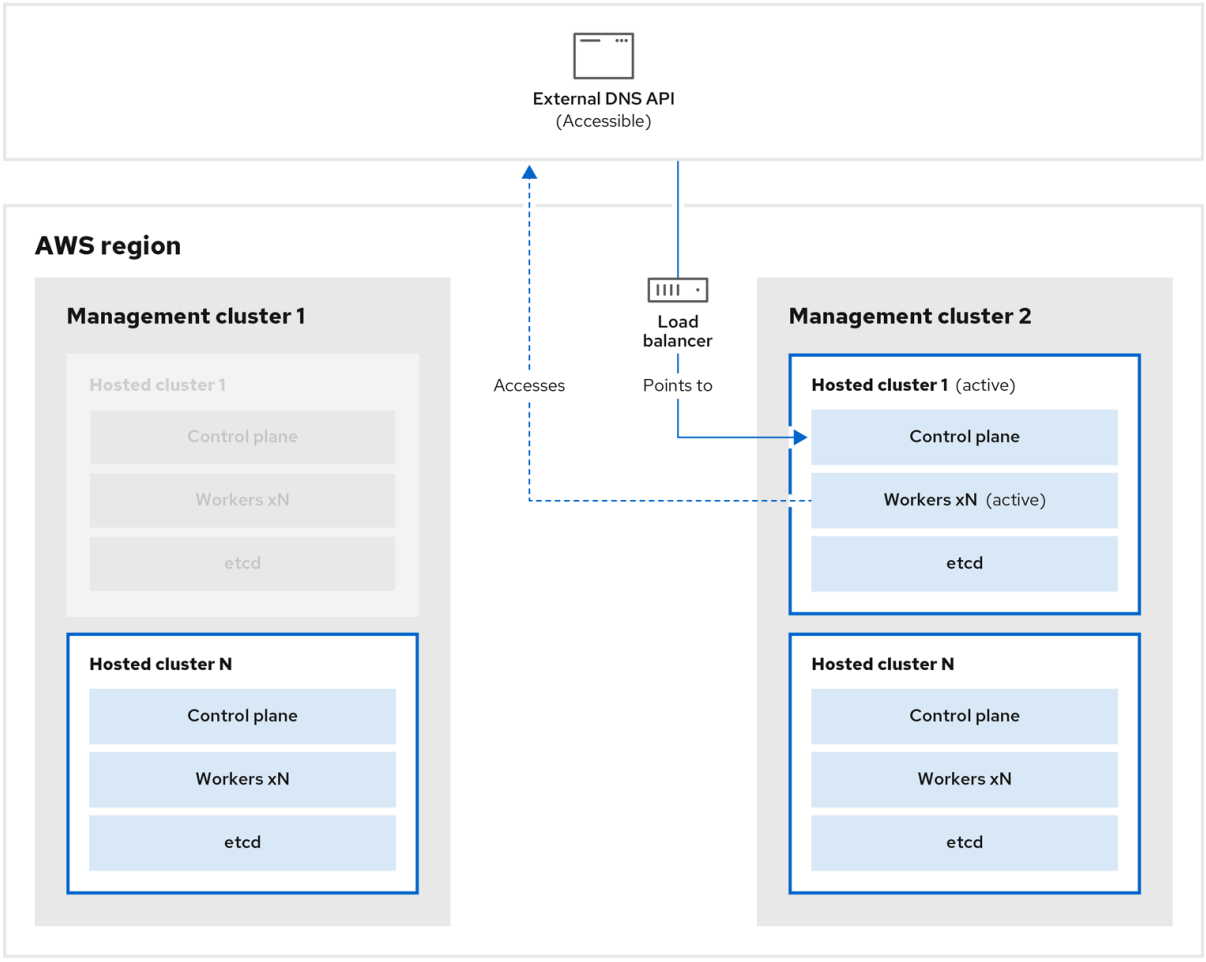
3. 管理クラスター 2 (宛先管理クラスターと見なすことができます) では、S3 バケットから etcd を復元し、ローカルマニフェストファイルからコントロールプレーンを復元します。このプロセスの間、外部 DNS API は停止し、ホストされたクラスター API にアクセスできなくなり、API を使用するワーカーはマニフェストファイルを更新できなくなりますが、ワークロードは引き続き実行されます。



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



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



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

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

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

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

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

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

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

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

# Hosted Control Plane
```

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

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

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

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

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

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

# MachineSets
echo "--> HostedCluster MachineSets:"
for s in $(oc get machineset -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} -o
name); do
   ms_name=$(echo ${s} | cut -f 2 -d /)
   oc get -n ${HC_CLUSTER_NS}-${HC_CLUSTER_NAME} $s -o yaml >
```

```
${BACKUP_DIR}/namespaces/${HC_CLUSTER_NS}-
${HC_CLUSTER_NAME}/machineset-${ms_name}.yaml
done

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

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

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

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

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

```
function clean_routes() {

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

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

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

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

   while [ ${ROUTES} -gt 2 ]
   do
      echo "Waiting for ExternalDNS Operator to clean the DNS Records in AWS Route53
where the zone id is: ${ZONE_ID}..."
      echo "Try: (${count}/${timeout})"
      sleep 10
      if [[ $count -eq timeout ]];then
         echo "Timeout waiting for cleaning the Route53 DNS records"
         exit 1
      fi
      count=$((count+1))
      ROUTES=$(aws route53 list-resource-record-sets --hosted-zone-id ${ZONE_ID} --max-
```

```
items 10000 --output json | grep -c ${EXTERNAL_DNS_DOMAIN})
    done
}

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

検証

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

次のステップ

ホストされたクラスターを復元します。

8.4.3. ホストされたクラスターの復元

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

前提条件

ソース管理クラスターからデータをバックアップしている。

ヒント

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

手順

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

for POD in ${ETCD_PODS}; do
  # Create a pre-signed URL for the etcd snapshot
  ETCD_SNAPSHOT="s3://${BUCKET_NAME}/${HC_CLUSTER_NAME}-${POD}-
snapshot.db"
  ETCD_SNAPSHOT_URL=$(AWS_DEFAULT_REGION=${MGMT2_REGION} aws s3
```

```
presign ${ETCD_SNAPSHOT})

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

cat ${HC_RESTORE_FILE}

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

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

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

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

検証

- ノードが完全に復元されたことを確認するには、次の関数を使用します。

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

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

次のステップ

クラスターをシャットダウンして削除します。

8.4.4. ホストされたクラスターのソース管理クラスターからの削除

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

前提条件

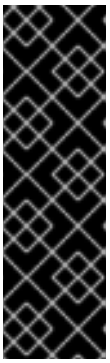
データをバックアップし、ソース管理クラスターに復元している。

ヒント

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

手順

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



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調整し、値を **Delete** に返すコントローラーが存在しないことを確認してください。これにより、データの損失が発生する可能性があります。

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

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

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

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

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

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

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

4. 次のコマンドを入力して、クラスターオブジェクトを削除します。

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

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

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

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

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

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

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

検証

- すべてが機能することを確認するには、次のコマンドを入力します。

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

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

# Inside the HostedCluster
export KUBECONFIG=${HC_KUBECONFIG}
oc get clusterversion
oc get nodes
```

次のステップ

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

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

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

第9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。

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

Hosted Control Plane クラスターの問題のトラブルシューティングが必要な場合は、**hypershift dump cluster** 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トされた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。
- **ホストされたクラスター**: このレベルの出力には、ホストされたクラスター内のすべてのリソースが含まれます。

ホストされた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。

```
$ hypershift dump cluster \
```

```
--name <hosted_cluster_name> \ 1
--namespace <hosted_cluster_namespace> \ 2
--dump-guest-cluster \
--artifact-dir clusterDump-<hosted_cluster_namespace>-<hosted_cluster_name>
```

1 ホストされたクラスターの名前を指定します。

2 ホストされたクラスターの namespace を指定します (例: **clusters**)。

出力例

```
2023-06-06T12:18:20+02:00  INFO  Archiving dump  {"command": "tar", "args": ["-cvzf",
"hypershift-dump.tar.gz", "cluster-scoped-resources", "event-filter.html", "namespaces",
"network_logs", "timestamp"]}
2023-06-06T12:18:21+02:00  INFO  Successfully archived dump  {"duration":
"1.519376292s"}
```

- ユーザー名またはサービスアカウントを使用して管理クラスターに対するすべてのクエリーを偽装するようにコマンドラインインターフェイスを設定するには、**--as** フラグを指定して **hypershift dump cluster** コマンドを入力します。
サービスアカウントには、namespace のすべてのオブジェクトをクエリーするための十分な権限が必要です。そのため、十分な権限があることを確認するために、**cluster-admin** ロールを使用することを推奨します。サービスアカウントは、**HostedControlPlane** リソースに存在するか、HostedControlPlane リソースの namespace をクエリーする権限を持っている必要があります。

  ユーザー名またはサービスアカウントに十分な権限がない場合、出力にはアクセス権限のあるオブジェクトのみが含まれます。そのプロセス中に、**forbidden** エラーが表示される場合があります。

  - サービスアカウントを使用して偽装を使用するには、次のコマンドを入力します。

    ```
    $ hypershift dump cluster \
      --name <hosted_cluster_name> \ 1
      --namespace <hosted_cluster_namespace> \ 2
      --dump-guest-cluster \
      --as "system:serviceaccount:<service_account_namespace>:
    <service_account_name>" \ 3
      --artifact-dir clusterDump-<hosted_cluster_namespace>-<hosted_cluster_name>
    ```

    1 ホストされたクラスターの名前を指定します。

    2 ホストされたクラスターの namespace を指定します (例: **clusters**)。

    3 **default** namespace と名前を指定します (例: **"system:serviceaccount:default:samplesa"**)。

  - ユーザー名を使用して偽装を使用するには、次のコマンドを入力します。

    ```
    $ hypershift dump cluster \
      --name <hosted_cluster_name> \ 1
      --namespace <hosted_cluster_namespace> \ 2
    ```

```
--dump-guest-cluster \
--as "<cluster_user_name>" \ 3
--artifact-dir clusterDump-<hosted_cluster_namespace>-<hosted_cluster_name>
```

1. ホストされたクラスターの名前を指定します。
2. ホストされたクラスターの namespace を指定します (例: **clusters**)。
3. クラスターのユーザー名を指定します (例: **cloud-admin**)。

9.2. 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)
```

検証

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。この例の **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。

次のコンポーネントが再起動されます。

- 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

9.3. ホストされたクラスターと HOSTED CONTROL PLANE の調整の一時停止

クラスターインスタンス管理者は、ホストされたクラスターと Hosted Control Plane の調整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、ホストされたクラスターまたは Hosted Control Plane の問題をデバッグする必要があるときは、調整を一時停止することができます。

手順

1. ホストされたクラスターと Hosted Control Plane の調整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。
   - 特定の時刻まで調整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、調整が一時停止します。

   - 調整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、調整は一時停止されます。

     HostedCluster リソースの一時停止調整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
```

9.4. 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、ホストされた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、ホストされた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> -o yaml --namespace
<HOSTED_CLUSTER_NAMESPACE>
```

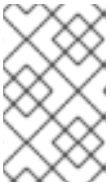
出力例

```
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) ホストされた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。

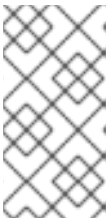


注記

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、ホストされた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、ホストされた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