自己管理型クラスタを追加します

Astra Control Service

NetApp
April 24, 2024

目次

自己管理型クラスタを追加します

Astra Control Serviceにパブリック自己管理型クラスタを追加

環境のセットアップが完了したら、 Kubernetes クラスタを作成し、 Astra Control Service に追加することができます。

自己管理型クラスタは、ユーザが直接プロビジョニングおよび管理するクラスタです。Astra Control Service は、パブリッククラウド環境で実行される自己管理型クラスタをサポートします。をアップロードすることで、Astra Control Serviceに自己管理型クラスタを追加できます `kubeconfig.yaml` ファイル。クラスタがここで説明する要件を満たしていることを確認する必要があります。

サポートされている**Kubernetes**ディストリビューション

Astra Control Serviceを使用して、次のタイプのパブリック自己管理クラスタを管理できます。

Kubernetesディストリビューション	サポートされるバージョン
Kubernetes（アップストリーム）	1.27〜1.29
Rancher Kubernetes Engine （ RKE ）	RKE 1：バージョン1.24.17、1.25.13、1.26.8（Rancher Manager 2.7.9を使用） RKE 2：Rancher Manager 2.6.13を使用したバージョン1.23.16および1.24.13 RKE 2：バージョン1.24.17、1.25.14、1.26.9（Rancher Manager 2.7.9を使用）
Red Hat OpenShift Container Platform	4.12~4.14

ここで説明する手順は、自己管理型クラスタがすでに作成されていることを前提としています。

- Astra Control Serviceにクラスタを追加
- [デフォルトのストレージクラスを変更する]

Astra Control Serviceにクラスタを追加

Astra Control Service にログインしたら、最初にクラスタの管理を開始します。Astra Control Serviceにクラスタを追加する前に、特定のタスクを実行し、クラスタが一定の要件を満たしていることを確認する必要があります。

作業を開始する前に

自己管理型クラスタは、ユーザが直接プロビジョニングおよび管理するクラスタです。Astra Control Serviceは、パブリッククラウド環境で実行される自己管理型クラスタをサポートします。自己管理型クラスタでは、Astra Control Provisionerを使用してNetAppストレージサービスを操作したり、Container Storage Interface（CSI）ドライバを使用してAmazon Elastic Block Store（EBS）、Azure Managed Disks、Google Persistent Diskを操作したりできます。

Astra Control Serviceは、次のKubernetesディストリビューションを使用する自己管理クラスタをサポートします。

- Red Hat OpenShift Container Platform
- Rancher Kubernetes Engineの略
- アップストリームKubernetes

自己管理型クラスタは、次の要件を満たしている必要があります。

- クラスタにインターネット経由でアクセスできる必要があります。
- CSIドライバで有効にしたストレージを使用または使用する予定の場合は、適切なCSIドライバをクラスタにインストールする必要があります。CSIドライバを使用してストレージを統合する方法の詳細については、ご使用のストレージサービスのマニュアルを参照してください。
- context要素を1つだけ含むcluster kubeconfigファイルにアクセスできる必要があります。をクリックします "以下の手順を参照して" kubeconfigファイルを生成します。
- プライベート認証局（CA）を参照するkubeconfigファイルを使用してクラスタを追加する場合は、`cluster` kubeconfigファイルのセクションを参照してください。これにより、Astra Controlでクラスタを追加できます。

  ```
  insecure-skip-tls-verify: true
  ```

- **rancher**のみ: Rancher環境でアプリケーションクラスタを管理する場合、rancherから提供されたkubeconfigファイルでアプリケーションクラスタのデフォルトコンテキストを変更して、rancher APIサーバコンテキストではなくコントロールプレーンコンテキストを使用します。これにより、Rancher API サーバの負荷が軽減され、パフォーマンスが向上します。
- * Astra Control Provisionerの要件*：クラスタを管理するには、Astra Tridentコンポーネントを含むAstra Control Provisionerを適切に設定する必要があります。
  - * Astra Trident環境要件の確認*：Astra Control Provisionerをインストールまたはアップグレードする前に、 "サポートされるフロントエンド、バックエンド、およびホスト構成"。
  - * Astra Control Provisioner機能を有効にする*：Astra Trident 23.10以降をインストールして有効にすることを強く推奨します。 "Astra Control Provisionerの高度なストレージ機能"。今後のリリースでは、Astra Control Provisionerが有効になっていない場合、Astra ControlはAstra Tridentをサポートしません。
  - ストレージバックエンドの構成:少なくとも1つのストレージバックエンドが "Astra Tridentで設定" クラスタのポリシーを確認してください。
  - ストレージクラスの設定：少なくとも1つのストレージクラスが "Astra Tridentで設定" クラスタのポリシーを確認してください。デフォルトのストレージクラスが設定されている場合は、デフォルトのアノテーションが設定されている*唯一の*ストレージクラスであることを確認します。

- ボリュームスナップショットコントローラを設定し、ボリュームスナップショットクラスをインストールする： "ボリュームSnapshotコントローラのインストール" Astra Controlでスナップショットを作成できるようにします。 "作成" 1つ以上 `VolumeSnapshotClass` Astra Tridentを使用：

手順

1. ダッシュボードで、 * Kubernetes クラスタの管理 * を選択します。

   プロンプトに従ってクラスタを追加します。

2. プロバイダ：*[その他]*タブを選択して、自己管理クラスタに関する詳細を追加します。

   a. その他：をアップロードして、自己管理クラスタに関する詳細を指定します `kubeconfig.yaml` ファイルまたはの内容を貼り付けます `kubeconfig.yaml` クリップボードからファイル。

   > 自分で作成する場合は `kubeconfig` ファイルには、* 1つの*コンテキストエレメントのみを定義する必要があります。を参照してください "Kubernetes のドキュメント" を参照してください `kubeconfig` ファイル。

3. クレデンシャル名：Astra Controlにアップロードする自己管理型クラスタのクレデンシャルの名前を指定します。デフォルトでは、クレデンシャル名がクラスタの名前として自動的に入力されます。
4. プライベートルート識別子：このフィールドはプライベートクラスタでのみ使用できます。
5. 「 * 次へ * 」を選択します。
6. （オプション）* Storage *：必要に応じて、このクラスタに導入されたKubernetesアプリケーションでデフォルトで使用するストレージクラスを選択します。

   a. クラスタの新しいデフォルトのストレージクラスを選択するには、*[新しいデフォルトのストレージクラスを割り当てる]*チェックボックスを有効にします。

   b. 新しいデフォルトのストレージクラスをリストから選択します。

   > 各クラウドプロバイダのストレージサービスには、コスト、パフォーマンス、耐障害性に関する次の情報が表示されます。
   >
   > - Cloud Volumes Service for Google Cloud：価格、パフォーマンス、耐障害性に関する情報
   > - Google Persistent Disk：コスト、パフォーマンス、耐障害性に関する情報は提供されません
   > - Azure NetApp Files ：パフォーマンスと耐障害性に関する情報
   > - Azure Managed Disks：価格、パフォーマンス、耐障害性に関する情報は提供されません
   > - Amazon Elastic Block Store：価格、パフォーマンス、耐障害性に関する情報がない
   > - Amazon FSX for NetApp ONTAP ：価格、パフォーマンス、耐障害性に関する情報は提供されません
   > - NetApp Cloud Volumes ONTAP ：価格、パフォーマンス、耐障害性に関する情報は提供されません

ストレージクラスごとに、次のいずれかのサービスを利用できます。

- "Cloud Volumes Service for Google Cloud"
- "Google Persistent Disk のことです"
  - "Azure NetApp Files の特長"
  - "Azure で管理されるディスク"
  - "Amazon Elastic Block Store"
  - "NetApp ONTAP 対応の Amazon FSX"
  - "NetApp Cloud Volumes ONTAP の略"

    の詳細を確認してください "Amazon Web Servicesクラスタのストレージクラス"。の詳細を確認してください "AKS クラスタのストレージクラス"。の詳細を確認してください "GKE クラスタのストレージクラス"。

c. 「 * 次へ * 」を選択します。

d. 確認と承認：構成の詳細を確認します。

e. [Add]*を選択して、Astra Control Serviceにクラスタを追加します。

デフォルトのストレージクラスを変更する

クラスタのデフォルトのストレージクラスは変更できます。

Astra Controlを使用してデフォルトのストレージクラスを変更する

クラスタのデフォルトのストレージクラスは、Astra Control内から変更できます。以前にインストールしたストレージバックエンドサービスをクラスタで使用している場合は、このメソッドを使用してデフォルトのストレージクラスを変更できない可能性があります（*デフォルトに設定*アクションは選択できません）。この場合は、を実行できます [コマンドラインを使用してデフォルトのストレージクラスを変更します]。

手順

1. Astra Control Service UI で、 [* Clusters] を選択します。
2. [* Clusters]ページで、変更するクラスタを選択します。
3. [* ストレージ *] タブを選択します。
4. 「ストレージクラス」カテゴリを選択します。
5. デフォルトとして設定するストレージクラスの* Actions *メニューを選択します。
6. 「デフォルトに設定」を選択します。

コマンドラインを使用してデフォルトのストレージクラスを変更します

Kubernetesコマンドを使用してクラスタのデフォルトのストレージクラスを変更することができます。この方法は、クラスタの構成に関係なく機能します。

手順

1. Kubernetesクラスタにログインします。

2. クラスタ内のストレージクラスを表示します。

```
kubectl get storageclass
```

3. デフォルトのストレージクラスからデフォルトの指定を削除する。<SC_NAME> をストレージクラスの名前に置き換えます。

```
kubectl patch storageclass <SC_NAME> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-
class":"false"}}}'
```

4. 別のストレージクラスをデフォルトとしてマークします。<SC_NAME> をストレージクラスの名前に置き換えます。

```
kubectl patch storageclass <SC_NAME> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

5. 新しいデフォルトストレージクラスを確認します。

```
kubectl get storageclass
```

Astra Control Serviceに自己管理型プライベートクラスタを追加

環境のセットアップが完了したら、 Kubernetes クラスタを作成し、 Astra Control Service に追加することができます。

自己管理型クラスタは、ユーザが直接プロビジョニングおよび管理するクラスタです。Astra Control Service は、パブリッククラウド環境で実行される自己管理型クラスタをサポートします。をアップロードすることで、Astra Control Serviceに自己管理型クラスタを追加できます `kubeconfig.yaml` ファイル。クラスタがここで説明する要件を満たしていることを確認する必要があります。

サポートされている**Kubernetes**ディストリビューション

Astra Control Serviceを使用して、次のタイプの自己管理型プライベートクラスタを管理できます。

Kubernetesディストリビューション	サポートされるバージョン
Kubernetes（アップストリーム）	1.27～1.29

Kubernetesディストリビューション	サポートされるバージョン
Rancher Kubernetes Engine （ RKE ）	RKE 1：バージョン1.24.17、1.25.13、1.26.8（Rancher Manager 2.7.9を使用） RKE 2：Rancher Manager 2.6.13を使用したバージョン1.23.16および1.24.13 RKE 2：バージョン1.24.17、1.25.14、1.26.9（Rancher Manager 2.7.9を使用）
Red Hat OpenShift Container Platform	4.12~4.14

ここで説明する手順は、すでにプライベートクラスタを作成し、リモートからアクセスするためのセキュアな方法を準備していることを前提としています。

Astra Control Serviceにプライベートクラスタを追加するには、次のタスクを実行する必要があります。

1. Astra Connectorのインストール
2. [永続的ストレージをセットアップする]
3. Astra Control Serviceに自己管理型プライベートクラスタを追加

Astra Connectorのインストール

プライベートクラスタを追加する前に、Astra Controlがクラスタと通信できるように、クラスタにAstra Connectorをインストールする必要があります。を参照してください "Kubernetesネイティブではないワークフローで管理されるプライベートクラスタ用に、以前のバージョンのAstra Connectorをインストール" 手順については、を参照し

永続的ストレージをセットアップする

クラスタに永続的ストレージを設定してください。永続的ストレージの設定の詳細については、『Get Started』ドキュメントを参照してください。

- "Azure NetApp Files を使用して Microsoft Azure をセットアップする"
- "Azure で管理されているディスクを使用して Microsoft Azure をセットアップする"
- "Amazon Web Servicesをセットアップする"
- "Google Cloud をセットアップします"

Astra Control Serviceに自己管理型プライベートクラスタを追加

プライベートクラスタをAstra Control Serviceに追加できるようになりました。

作業を開始する前に

自己管理型クラスタは、ユーザが直接プロビジョニングおよび管理するクラスタです。Astra Control Serviceは、パブリッククラウド環境で実行される自己管理型クラスタをサポートします。自己管理型クラスタでは、Astra Control Provisionerを使用してNetAppストレージサービスを操作したり、Container Storage Interface（CSI）ドライバを使用してAmazon Elastic Block Store（EBS）、Azure Managed Disks、Google Persistent Diskを操作したりできます。

Astra Control Serviceは、次のKubernetesディストリビューションを使用する自己管理クラスタをサポートします。

- Red Hat OpenShift Container Platform
- Rancher Kubernetes Engineの略
- アップストリームKubernetes

自己管理型クラスタは、次の要件を満たしている必要があります。

- クラスタにインターネット経由でアクセスできる必要があります。
- CSIドライバで有効にしたストレージを使用または使用する予定の場合は、適切なCSIドライバをクラスタにインストールする必要があります。CSIドライバを使用してストレージを統合する方法の詳細については、ご使用のストレージサービスのマニュアルを参照してください。
- context要素を1つだけ含むcluster kubeconfigファイルにアクセスできる必要があります。をクリックします "以下の手順を参照して" kubeconfigファイルを生成します。
- プライベート認証局（CA）を参照するkubeconfigファイルを使用してクラスタを追加する場合は、`cluster` kubeconfigファイルのセクションを参照してください。これにより、Astra Controlでクラスタを追加できます。

  ```
  insecure-skip-tls-verify: true
  ```

- **rancher**のみ: Rancher環境でアプリケーションクラスタを管理する場合、rancherから提供されたkubeconfigファイルでアプリケーションクラスタのデフォルトコンテキストを変更して、rancher APIサーバコンテキストではなくコントロールプレーンコンテキストを使用します。これにより、Rancher API サーバの負荷が軽減され、パフォーマンスが向上します。
- * Astra Control Provisionerの要件*：クラスタを管理するには、Astra Tridentコンポーネントを含むAstra Control Provisionerを適切に設定する必要があります。
  - * Astra Trident環境要件の確認*：Astra Control Provisionerをインストールまたはアップグレードする前に、 "サポートされるフロントエンド、バックエンド、およびホスト構成"。
  - * Astra Control Provisioner機能を有効にする*：Astra Trident 23.10以降をインストールして有効にすることを強く推奨します。 "Astra Control Provisionerの高度なストレージ機能"。今後のリリースでは、Astra Control Provisionerが有効になっていない場合、Astra ControlはAstra Tridentをサポートしません。
  - ストレージバックエンドの構成:少なくとも1つのストレージバックエンドが "Astra Tridentで設定" クラスタのポリシーを確認してください。
  - ストレージクラスの設定：少なくとも1つのストレージクラスが "Astra Tridentで設定" クラスタのポリシーを確認してください。デフォルトのストレージクラスが設定されている場合は、デフォルトのアノテーションが設定されている*唯一の*ストレージクラスであることを確認します。

- ボリュームスナップショットコントローラを設定し、ボリュームスナップショットクラスをインストールする： "ボリュームSnapshotコントローラのインストール" Astra Controlでスナップショットを作成できるようにします。 "作成" 1つ以上 `VolumeSnapshotClass` Astra Tridentを使用：

手順

1. ダッシュボードで、 * Kubernetes クラスタの管理 * を選択します。

   プロンプトに従ってクラスタを追加します。

2. プロバイダ：*[その他]*タブを選択して、自己管理クラスタに関する詳細を追加します。
3. その他：をアップロードして、自己管理クラスタに関する詳細を指定します `kubeconfig.yaml` ファイルまたはの内容を貼り付けます `kubeconfig.yaml` クリップボードからファイル。

   > 自分で作成する場合は `kubeconfig` ファイルには、* 1つの*コンテキストエレメントのみを定義する必要があります。を参照してください "以下の手順を参照して" を参照してください `kubeconfig` ファイル。

4. クレデンシャル名：Astra Controlにアップロードする自己管理型クラスタのクレデンシャルの名前を指定します。デフォルトでは、クレデンシャル名がクラスタの名前として自動的に入力されます。
5. プライベートルート識別子：Astra Connectorから取得できるプライベートルート識別子を入力します。を使用してAstra Connectorを照会した場合 `kubectl get astraconnector -n astra-connector` プライベートルート識別子はと呼ばれます `ASTRACONNECTORID`。

   > プライベートルート識別子は、AstraでプライベートKubernetesクラスタを管理できるようにするAstra Connectorに関連付けられた名前です。この場合、プライベートクラスタは、APIサーバをインターネットに公開しないKubernetesクラスタです。

6. 「 * 次へ * 」を選択します。
7. （オプション）* Storage *：必要に応じて、このクラスタに導入されたKubernetesアプリケーションでデフォルトで使用するストレージクラスを選択します。
   a. クラスタの新しいデフォルトのストレージクラスを選択するには、*[新しいデフォルトのストレージクラスを割り当てる]*チェックボックスを有効にします。
   b. 新しいデフォルトのストレージクラスをリストから選択します。

各クラウドプロバイダのストレージサービスには、コスト、パフォーマンス、耐障害性に関する次の情報が表示されます。

- Cloud Volumes Service for Google Cloud：価格、パフォーマンス、耐障害性に関する情報
- Google Persistent Disk：コスト、パフォーマンス、耐障害性に関する情報は提供されません
- Azure NetApp Files ：パフォーマンスと耐障害性に関する情報
- Azure Managed Disks：価格、パフォーマンス、耐障害性に関する情報は提供されません
- Amazon Elastic Block Store：価格、パフォーマンス、耐障害性に関する情報がない
- Amazon FSX for NetApp ONTAP ：価格、パフォーマンス、耐障害性に関する情報は提供されません
- NetApp Cloud Volumes ONTAP ：価格、パフォーマンス、耐障害性に関する情報は提供されません

ストレージクラスごとに、次のいずれかのサービスを利用できます。

- "Cloud Volumes Service for Google Cloud"
- "Google Persistent Disk のことです"
- "Azure NetApp Files の特長"
- "Azure で管理されるディスク"
- "Amazon Elastic Block Store"
- "NetApp ONTAP 対応の Amazon FSX"
- "NetApp Cloud Volumes ONTAP の略"

  の詳細を確認してください "Amazon Web Servicesクラスタのストレージクラス"。の詳細を確認してください "AKS クラスタのストレージクラス"。の詳細を確認してください "GKE クラスタのストレージクラス"。

c. 「 * 次へ * 」を選択します。

d. 確認と承認：構成の詳細を確認します。

e. [Add]*を選択して、Astra Control Serviceにクラスタを追加します。

デフォルトのストレージクラスを変更する

クラスタのデフォルトのストレージクラスは変更できます。

Astra Controlを使用してデフォルトのストレージクラスを変更する

クラスタのデフォルトのストレージクラスは、Astra Control内から変更できます。以前にインストールしたストレージバックエンドサービスをクラスタで使用している場合は、このメソッドを使用してデフォルトのストレージクラスを変更できない可能性があります（*デフォルトに設定*アクションは選択できません）。この場合は、を実行できます [コマンドラインを使用してデフォルトのストレージクラスを変更します]。

手順

1. Astra Control Service UI で、 [* Clusters] を選択します。
2. [* Clusters]ページで、変更するクラスタを選択します。
3. [* ストレージ *] タブを選択します。
4. 「ストレージクラス」カテゴリを選択します。
5. デフォルトとして設定するストレージクラスの* Actions *メニューを選択します。
6. 「デフォルトに設定」を選択します。

コマンドラインを使用してデフォルトのストレージクラスを変更します

Kubernetesコマンドを使用してクラスタのデフォルトのストレージクラスを変更することができます。この方法は、クラスタの構成に関係なく機能します。

手順

1. Kubernetesクラスタにログインします。
2. クラスタ内のストレージクラスを表示します。

```
kubectl get storageclass
```

3. デフォルトのストレージクラスからデフォルトの指定を削除する。<SC_NAME> をストレージクラスの名前に置き換えます。

```
kubectl patch storageclass <SC_NAME> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-
class":"false"}}}'
```

4. 別のストレージクラスをデフォルトとしてマークします。<SC_NAME> をストレージクラスの名前に置き換えます。

```
kubectl patch storageclass <SC_NAME> -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

5. 新しいデフォルトストレージクラスを確認します。

```
kubectl get storageclass
```

Astra Tridentのバージョンを確認

ストレージサービスにAstra Control ProvisionerまたはAstra Tridentを使用する自己管理型クラスタを追加するには、Astra Tridentのバージョンが23.10以降であることを確認し

てください。

手順

1. 実行しているAstra Tridentのバージョンを確認します。

```
kubectl get tridentversions -n trident
```

Astra Tridentがインストールされている場合は、次のような出力が表示されます。

```
NAME      VERSION
trident   24.02.0
```

Astra Tridentがインストールされていない場合は、次のような出力が表示されます。

```
error: the server doesn't have a resource type "tridentversions"
```

2. 次のいずれかを実行します。
   - Astra Trident 23.01以前を実行している場合は、以下を使用 "手順" Astra Control Provisionerにアップグレードする前に、Astra Tridentの最新バージョンにアップグレードすること。可能です "直接アップグレードを実行する" Astra Tridentがバージョン24.02の4リリース期間内にある場合は、Astra Control Provisioner 24.02をダウンロードします。たとえば、Astra Trident 23.04からAstra Control Provisioner 24.02に直接アップグレードできます。
   - Astra Trident 23.10以降を実行している場合は、Astra Control Provisionerが "有効"。Astra Control Provisionerは、23.10より前のリリースのAstra Control Centerでは機能しません。 "Astra Control Provisionerのアップグレード" 最新の機能にアクセスするには、アップグレードするAstra Control Centerと同じバージョンを使用する必要があります。
3. ポッドが実行されていることを確認します。

```
kubectl get pods -n trident
```

4. サポートされているAstra Tridentドライバをストレージクラスで使用しているかどうかを確認します。プロビジョニング担当者の名前はとします `csi.trident.netapp.io`。次の例を参照してください。

```
kubectl get sc
```

回答例：

```
NAME                    PROVISIONER                    RECLAIMPOLICY
VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
ontap-gold (default)   csi.trident.netapp.io          Delete
Immediate           true                   5d23h
```

kubeconfigファイルを作成します

kubeconfigファイルを使用して、Astra Control Serviceにクラスタを追加できます。追加するクラスタのタイプによっては、特定の手順を使用してクラスタ用のkubeconfigファイルを手動で作成しなければならない場合があります。

- Amazon EKSクラスタ用のkubeconfigファイルを作成します
- Red Hat OpenShift Service on AWS（ROSA）クラスタ用のkubeconfigファイルを作成する
- [他のタイプのクラスタ用にkubeconfigファイルを作成します]

Amazon EKSクラスタ用のkubeconfigファイルを作成します

以下の手順に従って、Amazon EKSクラスタ用のkubeconfigファイルと永続的トークンシークレットを作成します。EKSでホストされるクラスタには、永続的なトークンシークレットが必要です。

手順

1. Amazonのドキュメントの手順に従って、kubeconfigファイルを生成します。

   "Amazon EKSクラスタ用のkubeconfigファイルを作成または更新します"

2. 次の手順でサービスアカウントを作成します。

   a. という名前のサービスアカウントファイルを作成します `astracontrol-service-account.yaml`。

      必要に応じてサービスアカウント名を調整します。ネームスペース `kube-system` これらの手順では必須です。ここでサービスアカウント名を変更する場合は、次の手順で同じ変更を適用する必要があります。

```
<strong>astracontrol-service-account.yaml</strong>
```

+

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: astra-admin-account
  namespace: kube-system
```

3. サービスアカウントを適用します。

```
kubectl apply -f astracontrol-service-account.yaml
```

4. を作成します `ClusterRoleBinding` という名前のファイルです `astracontrol-clusterrolebinding.yaml`。

```
<strong>astracontrol-clusterrolebinding.yaml</strong>
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: astra-admin-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: astra-admin-account
  namespace: kube-system
```

5. クラスタロールバインドを適用します。

```
kubectl apply -f astracontrol-clusterrolebinding.yaml
```

6. という名前のサービスアカウントトークンシークレットファイルを作成します `astracontrol-secret.yaml`。

```
<strong>astracontrol-secret.yaml</strong>
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: astra-admin-account
  name: astra-admin-account
  namespace: kube-system
type: kubernetes.io/service-account-token
```

7. トークンシークレットを適用します。

```
kubectl apply -f astracontrol-secret.yaml
```

8. トークンシークレットを取得します。

```
kubectl get secret astra-admin-account -n kube-system -o
jsonpath='{.data.token}' | base64 -d
```

9. を交換します `user` 次の例に示すように、AWS EKS kubeconfigファイルのセクションでトークンを指定します。

```
user:
    token: k8s-aws-
v1.aHR0cHM6Ly9zdHMudXMtd2VzdC0yLmFtYXpvbmF3cy5jb20vP0FjdGlvbj1HZXRDYWxsZ
XJJZGVudGl0eSZWZXJzaW9uPTIwMTEtMDYtMTUmWC1BbXotQWxnb3JpdGhtPUFXUzQtSE1BQ
y1TSEEyNTYmWC1BbXotQ3JlZGVudGlhbD1BS0lBM1JEWDdKU0haWU9LSEQ2SyUyRjIwMjMwN
DAzJTJGdXMtd2VzdC0yJTJGc3RzJTJGYXdzNF9yZXF1ZXN0JlgtQW16LURhdGU9MjAyMzA0M
DNUMjA0MzQwWiZYLUFtei1FeHBpcmVzPTYwJlgtQW16LVNpZ25lZEhlYWRlcnM9aG9zdCUzQ
ngtazhzLWF3cy1pZCZYLUFtei1TaWduYXR1cmU9YjU4ZWM0NzdiM2NkZGYxNGRhNzU4MGI2Z
WQ2zY2NzI2YWIwM2UyNThjMjRhNTJjNmVhNjc4MTRlNjJkOTg2Mg
```

Red Hat OpenShift Service on AWS（ROSA）クラスタ用のkubeconfigファイルを作成する

次の手順に従って、Red Hat OpenShift Service on AWS（ROSA）クラスタ用のkubeconfigファイルを作成します。

手順

1. ROSAクラスタにログインします。
2. サービスアカウントを作成します。

```
oc create sa astracontrol-service-account
```

3. クラスタロールを追加します。

```
oc adm policy add-cluster-role-to-user cluster-admin -z astracontrol-
service-account
```

4. 次の例を使用して、サービスアカウントシークレットコンフィギュレーションファイルを作成します。

```
<strong>secret-astra-sa.yaml</strong>
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: secret-astracontrol-service-account
  annotations:
    kubernetes.io/service-account.name: "astracontrol-service-account"
type: kubernetes.io/service-account-token
```

5. シークレットを作成します。

```
oc create -f secret-astra-sa.yaml
```

6. 作成したサービスアカウントを編集し、Astra Controlサービスアカウントのシークレット名を `secrets` セクション。

```
oc edit sa astracontrol-service-account
```

```yaml
apiVersion: v1
imagePullSecrets:
- name: astracontrol-service-account-dockercfg-dvfcd
kind: ServiceAccount
metadata:
  creationTimestamp: "2023-08-04T04:18:30Z"
  name: astracontrol-service-account
  namespace: default
  resourceVersion: "169770"
  uid: 965fa151-923f-4fbd-9289-30cad15998ac
secrets:
- name: astracontrol-service-account-dockercfg-dvfcd
- name: secret-astracontrol-service-account ####ADD THIS ONLY####
```

7. サービスアカウントのシークレットを一覧表示します（置き換えます） `<CONTEXT>` インストールに適したコンテキストを使用して、次の操作を行います。

```
kubectl get serviceaccount astracontrol-service-account --context
<CONTEXT> --namespace default -o json
```

出力の末尾は次のようになります。

```
"secrets": [
{ "name": "astracontrol-service-account-dockercfg-dvfcd"},
{ "name": "secret-astracontrol-service-account"}
]
```

内の各要素のインデックス `secrets` アレイは0から始まります。上記の例では、のインデックスです `astracontrol-service-account-dockercfg-dvfcd` は0、のインデックスです `secret-astracontrol-service-account` は1です。出力で、サービスアカウントシークレットのインデックス番号をメモします。このインデックス番号は次の手順で必要になります。

8. 次のように kubeconfig を生成します。
   a. を作成します `create-kubeconfig.sh` ファイル。交換してください `TOKEN_INDEX` 次のスクリプトの先頭に正しい値を入力します。

```
<strong>create-kubeconfig.sh</strong>
```

```
# Update these to match your environment.
# Replace TOKEN_INDEX with the correct value
# from the output in the previous step. If you
# didn't change anything else above, don't change
# anything else here.

SERVICE_ACCOUNT_NAME=astracontrol-service-account
NAMESPACE=default
NEW_CONTEXT=astracontrol
KUBECONFIG_FILE='kubeconfig-sa'

CONTEXT=$(kubectl config current-context)

SECRET_NAME=$(kubectl get serviceaccount ${SERVICE_ACCOUNT_NAME} \
  --context ${CONTEXT} \
  --namespace ${NAMESPACE} \
  -o jsonpath='{.secrets[TOKEN_INDEX].name}')
TOKEN_DATA=$(kubectl get secret ${SECRET_NAME} \
  --context ${CONTEXT} \
  --namespace ${NAMESPACE} \
  -o jsonpath='{.data.token}')

TOKEN=$(echo ${TOKEN_DATA} | base64 -d)

# Create dedicated kubeconfig
# Create a full copy
kubectl config view --raw > ${KUBECONFIG_FILE}.full.tmp
```

```
# Switch working context to correct context
kubectl --kubeconfig ${KUBECONFIG_FILE}.full.tmp config use-context
${CONTEXT}

# Minify
kubectl --kubeconfig ${KUBECONFIG_FILE}.full.tmp \
  config view --flatten --minify > ${KUBECONFIG_FILE}.tmp

# Rename context
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  rename-context ${CONTEXT} ${NEW_CONTEXT}

# Create token user
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  set-credentials ${CONTEXT}-${NAMESPACE}-token-user \
  --token ${TOKEN}

# Set context to use token user
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  set-context ${NEW_CONTEXT} --user ${CONTEXT}-${NAMESPACE}-token
-user

# Set context to correct namespace
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  set-context ${NEW_CONTEXT} --namespace ${NAMESPACE}

# Flatten/minify kubeconfig
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  view --flatten --minify > ${KUBECONFIG_FILE}

# Remove tmp
rm ${KUBECONFIG_FILE}.full.tmp
rm ${KUBECONFIG_FILE}.tmp
```

b. コマンドをソースにし、 Kubernetes クラスタに適用します。

```
source create-kubeconfig.sh
```

9. （オプション）クラスタにわかりやすい名前にコバーベキューの名前を変更します。

```
mv kubeconfig-sa YOUR_CLUSTER_NAME_kubeconfig
```

他のタイプのクラスタ用に**kubeconfig**ファイルを作成します

以下の手順に従って、Rancher、Upstream Kubernetes、およびRed Hat OpenShiftクラスタ用に、制限付きまたは拡張されたロールkubeconfigファイルを作成します。

kubeconfigを使用して管理されるクラスタについては、必要に応じて、Astra Control Service用の制限された権限または拡張された権限管理者ロールを作成できます。

この手順を使用すると、次のいずれかのシナリオで環境を環境化する場合に、別のkubeconfigを作成できます。

- 管理対象のクラスタに対するAstra Controlの権限を制限する
- 複数のコンテキストを使用し、インストール時に設定されたデフォルトのAstra Control kubeconfigは使用できません。また、単一のコンテキストを持つ限定されたロールは環境では機能しません。

作業を開始する前に

手順 の手順を実行する前に、管理するクラスタに次の情報があることを確認してください。

- A "サポートされているバージョン" のkubectlがインストールされています。
- Astra Control Serviceを使用して追加および管理するクラスタへのkubectlアクセス

> この手順では、Astra Controlサービスを実行しているクラスタへのkubectlアクセスは必要ありません。

- アクティブなコンテキストのクラスタ管理者の権限で管理するクラスタのアクティブなkubeconfigです

手順

1. サービスアカウントを作成します。
   a. という名前のサービスアカウントファイルを作成します `astracontrol-service-account.yaml`。

```
<strong>astracontrol-service-account.yaml</strong>
```

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: astracontrol-service-account
  namespace: default
```

   b. サービスアカウントを適用します。

```
kubectl apply -f astracontrol-service-account.yaml
```

2. 次のいずれかのクラスタロールを作成し、Astra Controlで管理するクラスタに必要な権限を割り当てます。

クラスタロールの制限

このロールには、Astra Controlでクラスタを管理するために必要な最小限の権限が含まれています。

a. を作成します `ClusterRole` という名前のファイル。例： `astra-admin-account.yaml`。

```
<strong>astra-admin-account.yaml</strong>
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: astra-admin-account
rules:

# Get, List, Create, and Update all resources
# Necessary to backup and restore all resources in an app
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - get
  - list
  - create
  - patch

# Delete Resources
# Necessary for in-place restore and AppMirror failover
- apiGroups:
  - ""
  - apps
  - autoscaling
  - batch
  - crd.projectcalico.org
  - extensions
  - networking.k8s.io
  - policy
  - rbac.authorization.k8s.io
  - snapshot.storage.k8s.io
  - trident.netapp.io
  resources:
  - configmaps
  - cronjobs
  - daemonsets
  - deployments
```

```
    - horizontalpodautoscalers
    - ingresses
    - jobs
    - namespaces
    - networkpolicies
    - persistentvolumeclaims
    - poddisruptionbudgets
    - pods
    - podtemplates
    - replicasets
    - replicationcontrollers
    - replicationcontrollers/scale
    - rolebindings
    - roles
    - secrets
    - serviceaccounts
    - services
    - statefulsets
    - tridentmirrorrelationships
    - tridentsnapshotinfos
    - volumesnapshots
    - volumesnapshotcontents
    verbs:
    - delete

  # Watch resources
  # Necessary to monitor progress
  - apiGroups:
    - ""
    resources:
    - pods
    - replicationcontrollers
    - replicationcontrollers/scale
    verbs:
    - watch

  # Update resources
  - apiGroups:
    - ""
    - build.openshift.io
    - image.openshift.io
    resources:
    - builds/details
    - replicationcontrollers
    - replicationcontrollers/scale
    - imagestreams/layers
```

```
  - imagestreamtags
  - imagetags
  verbs:
  - update
```

b. （OpenShiftクラスタの場合のみ） `astra-admin-account.yaml` ファイル：

```
# OpenShift security
- apiGroups:
  - security.openshift.io
  resources:
  - securitycontextconstraints
  verbs:
  - use
  - update
```

c. クラスタロールを適用します。

```
kubectl apply -f astra-admin-account.yaml
```

クラスタロールの拡張

このロールには、Astra Controlで管理するクラスタに対する権限が拡張されています。このロールは、複数のコンテキストを使用し、インストール時に設定されたデフォルトのAstra Control kubeconfigを使用できない場合や、単一のコンテキストを持つ限定されたロールが環境で機能しない場合に使用できます。

> 次のようになります `ClusterRole` 手順はKubernetesの一般的な例です。ご使用の環境に固有の手順については、ご使用のKubernetesディストリビューションのドキュメントを参照してください。

a. を作成します `ClusterRole` という名前のファイル。例： `astra-admin-account.yaml`。

```
<strong>astra-admin-account.yaml</strong>
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: astra-admin-account
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- nonResourceURLs:
  - '*'
  verbs:
  - '*'
```

b. クラスタロールを適用します。

```
kubectl apply -f astra-admin-account.yaml
```

3. サービスアカウントへのクラスタロールバインド用に、クラスタロールを作成します。

a. を作成します `ClusterRoleBinding` という名前のファイルです `astracontrol-clusterrolebinding.yaml`。

```
<strong>astracontrol-clusterrolebinding.yaml</strong>
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: astracontrol-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: astra-admin-account
subjects:
- kind: ServiceAccount
  name: astracontrol-service-account
  namespace: default
```

b. クラスタロールバインドを適用します。

```
kubectl apply -f astracontrol-clusterrolebinding.yaml
```

4. トークンシークレットを作成して適用します。
   a. という名前のトークンシークレットファイルを作成します。 `secret-astracontrol-service-account.yaml`。

```
<strong>secret-astracontrol-service-account.yaml</strong>
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: secret-astracontrol-service-account
  namespace: default
  annotations:
    kubernetes.io/service-account.name: "astracontrol-service-
account"
type: kubernetes.io/service-account-token
```

   b. トークンシークレットを適用します。

```
kubectl apply -f secret-astracontrol-service-account.yaml
```

5. トークンシークレットの名前を `secrets` Array（次の例の最後の行）：

```
kubectl edit sa astracontrol-service-account
```

```
apiVersion: v1
imagePullSecrets:
- name: astracontrol-service-account-dockercfg-48xhx
kind: ServiceAccount
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"v1","kind":"ServiceAccount","metadata":{"annotations":{},
"name":"astracontrol-service-account","namespace":"default"}}
  creationTimestamp: "2023-06-14T15:25:45Z"
  name: astracontrol-service-account
  namespace: default
  resourceVersion: "2767069"
  uid: 2ce068c4-810e-4a96-ada3-49cbf9ec3f89
secrets:
- name: astracontrol-service-account-dockercfg-48xhx
<strong>- name: secret-astracontrol-service-account</strong>
```

6. サービスアカウントのシークレットを一覧表示します（置き換えます） `<context>` インストールに適したコンテキストを使用して、次の操作を行います。

```
kubectl get serviceaccount astracontrol-service-account --context
<context> --namespace default -o json
```

出力の末尾は次のようになります。

```
"secrets": [
{ "name": "astracontrol-service-account-dockercfg-48xhx"},
{ "name": "secret-astracontrol-service-account"}
]
```

内の各要素のインデックス `secrets` アレイは0から始まります。上記の例では、のインデックスです `astracontrol-service-account-dockercfg-48xhx` は0、のインデックスです `secret-astracontrol-service-account` は1です。出力で、サービスアカウントシークレットのインデックス番号をメモします。このインデックス番号は次の手順で必要になります。

7. 次のように kubeconfig を生成します。
   a. を作成します `create-kubeconfig.sh` ファイル。
   b. 交換してください `TOKEN_INDEX` 次のスクリプトの先頭に正しい値を入力します。

```
<strong>create-kubeconfig.sh</strong>
```

```
# Update these to match your environment.
# Replace TOKEN_INDEX with the correct value
# from the output in the previous step. If you
# didn't change anything else above, don't change
# anything else here.

SERVICE_ACCOUNT_NAME=astracontrol-service-account
NAMESPACE=default
NEW_CONTEXT=astracontrol
KUBECONFIG_FILE='kubeconfig-sa'

CONTEXT=$(kubectl config current-context)

SECRET_NAME=$(kubectl get serviceaccount ${SERVICE_ACCOUNT_NAME} \
  --context ${CONTEXT} \
  --namespace ${NAMESPACE} \
  *-o jsonpath='{.secrets[TOKEN_INDEX].name}')
TOKEN_DATA=$(kubectl get secret ${SECRET_NAME} \
  --context ${CONTEXT} \
  --namespace ${NAMESPACE} \
  -o jsonpath='{.data.token}')

TOKEN=$(echo ${TOKEN_DATA} | base64 -d)

# Create dedicated kubeconfig
# Create a full copy
kubectl config view --raw > ${KUBECONFIG_FILE}.full.tmp

# Switch working context to correct context
kubectl --kubeconfig ${KUBECONFIG_FILE}.full.tmp config use-context
${CONTEXT}

# Minify
kubectl --kubeconfig ${KUBECONFIG_FILE}.full.tmp \
  config view --flatten --minify > ${KUBECONFIG_FILE}.tmp

# Rename context
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  rename-context ${CONTEXT} ${NEW_CONTEXT}

# Create token user
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  set-credentials ${CONTEXT}-${NAMESPACE}-token-user \
  --token ${TOKEN}

# Set context to use token user
```

```
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  set-context ${NEW_CONTEXT} --user ${CONTEXT}-${NAMESPACE}-token-
user

# Set context to correct namespace
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  set-context ${NEW_CONTEXT} --namespace ${NAMESPACE}

# Flatten/minify kubeconfig
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  view --flatten --minify > ${KUBECONFIG_FILE}

# Remove tmp
rm ${KUBECONFIG_FILE}.full.tmp
rm ${KUBECONFIG_FILE}.tmp
```

c. コマンドをソースにし、 Kubernetes クラスタに適用します。

```
source create-kubeconfig.sh
```

8. （オプション）クラスタにわかりやすい名前にコバーベキューの名前を変更します。

```
mv kubeconfig-sa YOUR_CLUSTER_NAME_kubeconfig
```