자가 관리 클러스터를 추가합니다

Astra Control Service

NetApp
April 24, 2024

목차

자가 관리 클러스터를 추가합니다

공용 자가 관리 클러스터를 **Astra Control Service**에 추가합니다

환경을 설정한 후에는 Kubernetes 클러스터를 생성하고 Astra Control Service에 추가할 준비가 된 것입니다.

자가 관리형 클러스터는 직접 프로비저닝하고 관리하는 클러스터입니다. Astra Control Service는 퍼블릭 클라우드 환경에서 실행되는 자체 관리형 클러스터를 지원합니다. A를 업로드하여 자가 관리 클러스터를 Astra Control Service에 추가할 수 있습니다 `kubeconfig.yaml` 파일. 클러스터가 여기에 설명된 요구 사항을 충족하는지 확인해야 합니다.

지원되는 **Kubernetes** 배포

Astra Control Service를 사용하여 다음과 같은 유형의 퍼블릭, 자가 관리 클러스터를 관리할 수 있습니다.

Kubernetes 배포	지원되는 버전
Kubernetes(업스트림)	1.27 ~ 1.29
RKE(Rancher Kubernetes Engine)	RKE 1: Rancher Manager 2.7.9 포함 버전 1.24.17, 1.25.13, 1.26.8 RKE 2: Rancher Manager 2.6.13이 있는 버전 1.23.16 및 1.24.13 RKE 2: Rancher Manager 2.7.9 포함 버전 1.24.17, 1.25.14, 1.26.9
Red Hat OpenShift Container Platform	4.12에서 4.14까지

이 지침에서는 이미 자체 관리형 클러스터를 생성했다고 가정합니다.

- 클러스터를 Astra Control Service에 추가합니다
- 기본 스토리지 클래스를 변경합니다

클러스터를 **Astra Control Service**에 추가합니다

Astra Control Service에 로그인한 후 첫 번째 단계는 클러스터 관리를 시작하는 것입니다. Astra Control Service에 클러스터를 추가하려면 먼저 특정 작업을 수행하고 클러스터가 특정 요구 사항을 충족하는지 확인해야 합니다.

자가 관리형 클러스터는 직접 프로비저닝하고 관리하는 클러스터입니다. Astra Control Service는 퍼블릭 클라우드 환경에서 실행되는 자체 관리형 클러스터를 지원합니다. 자체 관리형 클러스터는 Astra Control Provisioner를 사용하여 NetApp 스토리지 서비스와 상호 연결하거나 컨테이너 스토리지 인터페이스(CSI) 드라이버를 사용하여 Amazon EBS(Elastic Block Store), Azure Managed Disks 및 Google Persistent Disk와 상호 작용할 수 있습니다.

Astra Control Service는 다음과 같은 Kubernetes 배포를 사용하는 자체 관리 클러스터를 지원합니다.

- Red Hat OpenShift Container Platform
- Rancher Kubernetes 엔진
- 업스트림 Kubernetes

자가 관리형 클러스터는 다음 요구사항을 충족해야 합니다.

- 클러스터를 인터넷을 통해 액세스할 수 있어야 합니다.
- CSI 드라이버로 활성화된 스토리지를 사용 중이거나 사용할 계획이면 해당 CSI 드라이버가 클러스터에 설치되어 있어야 합니다. CSI 드라이버를 사용하여 스토리지를 통합하는 방법에 대한 자세한 내용은 스토리지 서비스 설명서를 참조하십시오.
- 하나의 컨텍스트 요소만 포함된 클러스터 kubecononfig 파일에 액세스할 수 있습니다. 를 따릅니다 "참조하십시오" kubecononfig 파일을 생성합니다.
- 개인 CA(인증 기관)를 참조하는 kubeconfig 파일을 사용하여 클러스터를 추가하는 경우 에 다음 줄을 추가합니다 `cluster` kubeconfig 파일의 섹션. 이를 통해 Astra Control이 클러스터를 추가할 수 있습니다.

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

- * Rancher 전용 *: Rancher 환경에서 애플리케이션 클러스터를 관리할 때 Rancher가 제공하는 kubecon무화과 파일에서 애플리케이션 클러스터의 기본 컨텍스트를 수정하여 Rancher API 서버 컨텍스트 대신 컨트롤 플레인 컨텍스트를 사용합니다. 따라서 Rancher API 서버의 부하가 줄어들고 성능이 향상됩니다.
- * Astra Control Provisioner 요구 사항 *: 클러스터를 관리하려면 Astra Trident 구성 요소를 포함하여 올바르게 구성된 Astra Control Provisioner가 있어야 합니다.
  - * Astra Trident 환경 요구 사항 검토 *: Astra Control Provisioner를 설치 또는 업그레이드하기 전에 를 검토하십시오 "지원되는 프런트엔드, 백엔드 및 호스트 구성".
  - * Astra Control Provisioner 기능 활성화 *: Astra Trident 23.10 이상을 설치하고 활성화하는 것이 좋습니다 "Astra Control Provisioner 고급 스토리지 기능". 향후 릴리즈에서 Astra Control Provisioner가 활성화되어 있지 않으면 Astra Control이 Astra Trident를 지원하지 않습니다.
  - * 스토리지 백엔드 구성 *: 최소 하나의 스토리지 백엔드가 있어야 합니다 "Astra Trident에서 구성됨" 클러스터에서.
  - * 스토리지 클래스 구성 *: 최소 하나의 스토리지 클래스가 있어야 합니다 "Astra Trident에서 구성됨" 클러스터에서. 기본 저장소 클래스가 구성된 경우 기본 주석이 있는 * 전용 * 저장소 클래스인지 확인합니다.
  - * 볼륨 스냅샷 컨트롤러 구성 및 볼륨 스냅샷 클래스 설치 *: "볼륨 스냅샷 컨트롤러를 설치합니다" 따라서 Astra Control에서 스냅샷을 생성할 수 있습니다. "생성" 하나 이상 `VolumeSnapshotClass` Astra

Trident 사용:

단계

1. 대시보드에서 * Kubernetes 클러스터 관리 * 를 선택합니다.

   표시되는 메시지에 따라 클러스터를 추가합니다.

2. * 공급자 *: * 기타 * 탭을 선택하여 자체 관리 클러스터에 대한 세부 정보를 추가합니다.

   a. * 기타 *: 을 업로드하여 자체 관리되는 클러스터에 대한 세부 정보를 제공합니다 `kubeconfig.yaml` 파일을 클릭하거나 의 내용을 붙여 넣습니다 `kubeconfig.yaml` 파일을 클립보드에 저장합니다.

   > 직접 만드는 경우 `kubeconfig` 파일에서 * 하나의 * 컨텍스트 요소만 정의해야 합니다. 을 참조하십시오 "Kubernetes 문서" 을 참조하십시오 `kubeconfig` 파일.

3. * 자격 증명 이름 *: Astra Control에 업로드하는 자가 관리 클러스터 자격 증명의 이름을 입력합니다. 기본적으로 자격 증명 이름은 클러스터 이름으로 자동 채워집니다.
4. * Private route identifier *: 이 필드는 전용 클러스터에서만 사용할 수 있습니다.
5. 다음 * 을 선택합니다.
6. (선택 사항) * 스토리지 *: 선택적으로 이 클러스터에 Kubernetes 애플리케이션을 배포할 스토리지 클래스를 선택하여 기본적으로 사용하도록 합니다.

   a. 클러스터에 대한 새 기본 스토리지 클래스를 선택하려면 * 새 기본 스토리지 클래스 할당 * 확인란을 설정합니다.

   b. 목록에서 새 기본 스토리지 클래스를 선택합니다.

   > 각 클라우드 공급자의 스토리지 서비스에는 다음과 같은 가격, 성능 및 복원력 정보가 표시됩니다.
   >
   > - Google Cloud용 Cloud Volumes Service: 가격, 성능 및 복원력 정보
   > - Google 영구 디스크: 가격, 성능 또는 복원력 정보를 사용할 수 없습니다
   > - Azure NetApp Files: 성능 및 복원력 정보
   > - Azure 관리 디스크: 사용 가능한 가격, 성능 또는 복원력 정보가 없습니다
   > - Amazon Elastic Block Store: 가격, 성능 또는 복원력 정보를 사용할 수 없습니다
   > - NetApp ONTAP용 Amazon FSx: 가격, 성능 또는 복원력 정보 없음
   > - NetApp Cloud Volumes ONTAP: 가격, 성능 또는 복원력 정보를 제공할 수 없습니다

   각 스토리지 클래스는 다음 서비스 중 하나를 활용할 수 있습니다.

   - "Google Cloud용 Cloud Volumes Service"
   - "Google 영구 디스크"
     - "Azure NetApp Files"
     - "Azure로 관리되는 디스크"
     - "Amazon Elastic Block Store를 클릭합니다"

- "NetApp ONTAP용 Amazon FSx"
- "NetApp Cloud Volumes ONTAP를 참조하십시오"

  에 대해 자세히 알아보십시오 "Amazon Web Services 클러스터용 스토리지 클래스입니다". 에 대해 자세히 알아보십시오 "AKS 클러스터용 스토리지 클래스입니다". 에 대해 자세히 알아보십시오 "GKE 클러스터용 저장소 클래스".

c. 다음 * 을 선택합니다.

d. * 검토 및 승인 *: 구성 세부 정보를 검토합니다.

e. 클러스터를 Astra Control Service에 추가하려면 * 추가 * 를 선택합니다.

기본 스토리지 클래스를 변경합니다

클러스터의 기본 스토리지 클래스를 변경할 수 있습니다.

Astra Control을 사용하여 기본 스토리지 클래스를 변경합니다

Astra Control 내에서 클러스터의 기본 스토리지 클래스를 변경할 수 있습니다. 클러스터에서 이전에 설치된 스토리지 백엔드 서비스를 사용하는 경우 이 방법을 사용하여 기본 스토리지 클래스를 변경하지 못할 수 있습니다(* 기본값으로 설정* 작업은 선택할 수 없음). 이 경우 를 사용할 수 있습니다 명령줄을 사용하여 기본 스토리지 클래스를 변경합니다.

단계

1. Astra Control Service UI에서 * Clusters * 를 선택합니다.
2. 클러스터 * 페이지에서 변경할 클러스터를 선택합니다.
3. Storage * 탭을 선택합니다.
4. 스토리지 클래스 * 범주를 선택합니다.
5. 기본값으로 설정할 스토리지 클래스에 대해 * Actions * 메뉴를 선택합니다.
6. Set as default * 를 선택합니다.

명령줄을 사용하여 기본 스토리지 클래스를 변경합니다

Kubernetes 명령을 사용하여 클러스터의 기본 스토리지 클래스를 변경할 수 있습니다. 이 방법은 클러스터의 구성에 관계없이 작동합니다.

단계

1. Kubernetes 클러스터에 로그인합니다.
2. 클러스터의 스토리지 클래스를 나열합니다.

```
kubectl get storageclass
```

3. 기본 스토리지 클래스에서 기본 지정을 제거합니다. <SC_NAME>를 스토리지 클래스 이름으로 바꿉니다.

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

4. 다른 스토리지 클래스를 기본값으로 표시합니다. <SC_NAME>를 스토리지 클래스 이름으로 바꿉니다.

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

5. 새 기본 스토리지 클래스를 확인합니다.

```
kubectl get storageclass
```

Astra Control Service에 프라이빗 자체 관리 클러스터를 추가합니다

환경을 설정한 후에는 Kubernetes 클러스터를 생성하고 Astra Control Service에 추가할 준비가 된 것입니다.

자가 관리형 클러스터는 직접 프로비저닝하고 관리하는 클러스터입니다. Astra Control Service는 퍼블릭 클라우드 환경에서 실행되는 자체 관리형 클러스터를 지원합니다. A를 업로드하여 자가 관리 클러스터를 Astra Control Service에 추가할 수 있습니다 `kubeconfig.yaml` 파일. 클러스터가 여기에 설명된 요구 사항을 충족하는지 확인해야 합니다.

지원되는 **Kubernetes** 배포

Astra Control Service를 사용하여 다음과 같은 유형의 프라이빗, 자가 관리 클러스터를 관리할 수 있습니다.

Kubernetes 배포	지원되는 버전
Kubernetes(업스트림)	1.27 ~ 1.29
RKE(Rancher Kubernetes Engine)	RKE 1: Rancher Manager 2.7.9 포함 버전 1.24.17, 1.25.13, 1.26.8 RKE 2: Rancher Manager 2.6.13이 있는 버전 1.23.16 및 1.24.13 RKE 2: Rancher Manager 2.7.9 포함 버전 1.24.17, 1.25.14, 1.26.9
Red Hat OpenShift Container Platform	4.12에서 4.14까지

이 지침은 이미 프라이빗 클러스터를 생성하고 원격 액세스를 위한 보안 방법을 준비했다고 가정합니다.

Astra Control Service에 프라이빗 클러스터를 추가하려면 다음 작업을 수행해야 합니다.

1. Astra Connector를 설치합니다

2. 영구 스토리지를 설정합니다
3. 프라이빗 자체 관리 클러스터를 Astra Control Service에 추가합니다

Astra Connector를 설치합니다

프라이빗 클러스터를 추가하기 전에 Astra Control이 클러스터와 통신할 수 있도록 클러스터에 Astra Connector를 설치해야 합니다. 을 참조하십시오 "Kubernetes가 아닌 네이티브 워크플로우로 관리되는 프라이빗 클러스터용 Astra Connector의 이전 버전을 설치합니다" 를 참조하십시오.

영구 스토리지를 설정합니다

클러스터의 영구 스토리지를 구성합니다. 영구 스토리지 구성에 대한 자세한 내용은 시작 설명서를 참조하십시오.

- "Azure NetApp Files를 사용하여 Microsoft Azure를 설정합니다"
- "Azure 관리 디스크를 사용하여 Microsoft Azure를 설정합니다"
- "Amazon Web Services를 설정합니다"
- "Google Cloud를 설정합니다"

프라이빗 자체 관리 클러스터를 **Astra Control Service**에 추가합니다

이제 Astra Control Service에 프라이빗 클러스터를 추가할 수 있습니다.

자가 관리형 클러스터는 직접 프로비저닝하고 관리하는 클러스터입니다. Astra Control Service는 퍼블릭 클라우드 환경에서 실행되는 자체 관리형 클러스터를 지원합니다. 자체 관리형 클러스터는 Astra Control Provisioner를 사용하여 NetApp 스토리지 서비스와 상호 연결하거나 컨테이너 스토리지 인터페이스(CSI) 드라이버를 사용하여 Amazon EBS(Elastic Block Store), Azure Managed Disks 및 Google Persistent Disk와 상호 작용할 수 있습니다.

Astra Control Service는 다음과 같은 Kubernetes 배포를 사용하는 자체 관리 클러스터를 지원합니다.

- Red Hat OpenShift Container Platform
- Rancher Kubernetes 엔진
- 업스트림 Kubernetes

자가 관리형 클러스터는 다음 요구사항을 충족해야 합니다.

- 클러스터를 인터넷을 통해 액세스할 수 있어야 합니다.
- CSI 드라이버로 활성화된 스토리지를 사용 중이거나 사용할 계획이면 해당 CSI 드라이버가 클러스터에 설치되어 있어야 합니다. CSI 드라이버를 사용하여 스토리지를 통합하는 방법에 대한 자세한 내용은 스토리지 서비스 설명서를 참조하십시오.
- 하나의 컨텍스트 요소만 포함된 클러스터 kubecononfig 파일에 액세스할 수 있습니다. 를 따릅니다 "참조하십시오" kubecononfig 파일을 생성합니다.
- 개인 CA(인증 기관)를 참조하는 kubeconfig 파일을 사용하여 클러스터를 추가하는 경우 에 다음 줄을 추가합니다 `cluster` kubeconfig 파일의 섹션. 이를 통해 Astra Control이 클러스터를 추가할 수 있습니다.

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

- * Rancher 전용 *: Rancher 환경에서 애플리케이션 클러스터를 관리할 때 Rancher가 제공하는 kubecon무화과 파일에서 애플리케이션 클러스터의 기본 컨텍스트를 수정하여 Rancher API 서버 컨텍스트 대신 컨트롤 플레인 컨텍스트를 사용합니다. 따라서 Rancher API 서버의 부하가 줄어들고 성능이 향상됩니다.
- * Astra Control Provisioner 요구 사항 *: 클러스터를 관리하려면 Astra Trident 구성 요소를 포함하여 올바르게 구성된 Astra Control Provisioner가 있어야 합니다.
  - * Astra Trident 환경 요구 사항 검토 *: Astra Control Provisioner를 설치 또는 업그레이드하기 전에 를 검토하십시오 "지원되는 프런트엔드, 백엔드 및 호스트 구성".
  - * Astra Control Provisioner 기능 활성화 *: Astra Trident 23.10 이상을 설치하고 활성화하는 것이 좋습니다 "Astra Control Provisioner 고급 스토리지 기능". 향후 릴리즈에서 Astra Control Provisioner가 활성화되어 있지 않으면 Astra Control이 Astra Trident를 지원하지 않습니다.
  - * 스토리지 백엔드 구성 *: 최소 하나의 스토리지 백엔드가 있어야 합니다 "Astra Trident에서 구성됨" 클러스터에서.
  - * 스토리지 클래스 구성 *: 최소 하나의 스토리지 클래스가 있어야 합니다 "Astra Trident에서 구성됨" 클러스터에서. 기본 저장소 클래스가 구성된 경우 기본 주석이 있는 * 전용 * 저장소 클래스인지 확인합니다.
  - * 볼륨 스냅샷 컨트롤러 구성 및 볼륨 스냅샷 클래스 설치 *: "볼륨 스냅샷 컨트롤러를 설치합니다" 따라서 Astra Control에서 스냅샷을 생성할 수 있습니다. "생성" 하나 이상 `VolumeSnapshotClass` Astra

Trident 사용:

단계

1. 대시보드에서 * Kubernetes 클러스터 관리 * 를 선택합니다.

   표시되는 메시지에 따라 클러스터를 추가합니다.

2. * 공급자 *: * 기타 * 탭을 선택하여 자체 관리 클러스터에 대한 세부 정보를 추가합니다.
3. * 기타 *: 을 업로드하여 자체 관리되는 클러스터에 대한 세부 정보를 제공합니다 `kubeconfig.yaml` 파일을 클릭하거나 의 내용을 붙여 넣습니다 `kubeconfig.yaml` 파일을 클립보드에 저장합니다.

   > 직접 만드는 경우 `kubeconfig` 파일에서 * 하나의 * 컨텍스트 요소만 정의해야 합니다. 을 참조하십시오 "참조하십시오" 을 참조하십시오 `kubeconfig` 파일.

4. * 자격 증명 이름 *: Astra Control에 업로드하는 자가 관리 클러스터 자격 증명의 이름을 입력합니다. 기본적으로 자격 증명 이름은 클러스터 이름으로 자동 채워집니다.
5. * Private route identifier *: Astra Connector로부터 얻을 수 있는 private route identifier를 입력합니다. 을 통해 Astra Connector에 문의하면 `kubectl get astraconnector -n astra-connector` 명령, 전용 라우트 식별자를 라고 합니다 `ASTRACONNECTORID`.

   > 프라이빗 경로 식별자는 Astra Connector와 연결된 이름으로, 프라이빗 Kubernetes 클러스터를 Astra에서 관리할 수 있도록 합니다. 이런 맥락에서 프라이빗 클러스터는 API 서버를 인터넷에 노출하지 않는 Kubernetes 클러스터입니다.

6. 다음 * 을 선택합니다.
7. (선택 사항) * 스토리지 *: 선택적으로 이 클러스터에 Kubernetes 애플리케이션을 배포할 스토리지 클래스를 선택하여 기본적으로 사용하도록 합니다.
   a. 클러스터에 대한 새 기본 스토리지 클래스를 선택하려면 * 새 기본 스토리지 클래스 할당 * 확인란을 설정합니다.
   b. 목록에서 새 기본 스토리지 클래스를 선택합니다.

   > 각 클라우드 공급자의 스토리지 서비스에는 다음과 같은 가격, 성능 및 복원력 정보가 표시됩니다.
   >
   > - Google Cloud용 Cloud Volumes Service: 가격, 성능 및 복원력 정보
   > - Google 영구 디스크: 가격, 성능 또는 복원력 정보를 사용할 수 없습니다
   > - Azure NetApp Files: 성능 및 복원력 정보
   > - Azure 관리 디스크: 사용 가능한 가격, 성능 또는 복원력 정보가 없습니다
   > - Amazon Elastic Block Store: 가격, 성능 또는 복원력 정보를 사용할 수 없습니다
   > - NetApp ONTAP용 Amazon FSx: 가격, 성능 또는 복원력 정보 없음
   > - NetApp Cloud Volumes ONTAP: 가격, 성능 또는 복원력 정보를 제공할 수 없습니다

   각 스토리지 클래스는 다음 서비스 중 하나를 활용할 수 있습니다.

- "Google Cloud용 Cloud Volumes Service"
- "Google 영구 디스크"
- "Azure NetApp Files"
- "Azure로 관리되는 디스크"
- "Amazon Elastic Block Store를 클릭합니다"
- "NetApp ONTAP용 Amazon FSx"
- "NetApp Cloud Volumes ONTAP를 참조하십시오"

  에 대해 자세히 알아보십시오 "Amazon Web Services 클러스터용 스토리지 클래스입니다". 에 대해 자세히 알아보십시오 "AKS 클러스터용 스토리지 클래스입니다". 에 대해 자세히 알아보십시오 "GKE 클러스터용 저장소 클래스".

c. 다음 * 을 선택합니다.

d. * 검토 및 승인 *: 구성 세부 정보를 검토합니다.

e. 클러스터를 Astra Control Service에 추가하려면 * 추가 * 를 선택합니다.

기본 스토리지 클래스를 변경합니다

클러스터의 기본 스토리지 클래스를 변경할 수 있습니다.

Astra Control을 사용하여 기본 스토리지 클래스를 변경합니다

Astra Control 내에서 클러스터의 기본 스토리지 클래스를 변경할 수 있습니다. 클러스터에서 이전에 설치된 스토리지 백엔드 서비스를 사용하는 경우 이 방법을 사용하여 기본 스토리지 클래스를 변경하지 못할 수 있습니다(* 기본값으로 설정* 작업은 선택할 수 없음). 이 경우 를 사용할 수 있습니다 명령줄을 사용하여 기본 스토리지 클래스를 변경합니다.

단계

1. Astra Control Service UI에서 * Clusters * 를 선택합니다.
2. 클러스터 * 페이지에서 변경할 클러스터를 선택합니다.
3. Storage * 탭을 선택합니다.
4. 스토리지 클래스 * 범주를 선택합니다.
5. 기본값으로 설정할 스토리지 클래스에 대해 * Actions * 메뉴를 선택합니다.
6. Set as default * 를 선택합니다.

명령줄을 사용하여 기본 스토리지 클래스를 변경합니다

Kubernetes 명령을 사용하여 클러스터의 기본 스토리지 클래스를 변경할 수 있습니다. 이 방법은 클러스터의 구성에 관계없이 작동합니다.

단계

1. Kubernetes 클러스터에 로그인합니다.
2. 클러스터의 스토리지 클래스를 나열합니다.

```
kubectl get storageclass
```

3. 기본 스토리지 클래스에서 기본 지정을 제거합니다. <SC_NAME>를 스토리지 클래스 이름으로 바꿉니다.

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

4. 다른 스토리지 클래스를 기본값으로 표시합니다. <SC_NAME>를 스토리지 클래스 이름으로 바꿉니다.

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

5. 새 기본 스토리지 클래스를 확인합니다.

```
kubectl get storageclass
```

Astra Trident 버전을 확인합니다

스토리지 서비스에 Astra Control Provisioner 또는 Astra Trident를 사용하는 자체 관리형 클러스터를 추가하려면 Astra Trident의 설치된 버전이 23.10 이상이어야 합니다.

단계

1. 실행 중인 Astra Trident 버전을 확인합니다.

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

Astra Trident가 설치된 경우 다음과 유사한 출력이 표시됩니다.

```
NAME      VERSION
trident   24.02.0
```

Astra Trident가 설치되지 않은 경우 다음과 유사한 출력이 표시됩니다.

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

2. 다음 중 하나를 수행합니다.

- Astra Trident 23.01 이하를 실행 중인 경우 다음을 사용합니다 "지침" Astra Control Provisioner로 업그레이드하기 전에 Astra Trident의 최신 버전으로 업그레이드하십시오. 가능합니다 "직접 업그레이드를 수행합니다" Astra Trident가 버전 24.02의 4개 릴리즈 윈도우 내에 있는 경우 Astra Control Provisioner 24.02에 등록됩니다. 예를 들어, Astra Trident 23.04에서 Astra Control Provisioner 24.02로 직접 업그레이드할 수 있습니다.
- Astra Trident 23.10 이상을 실행 중인 경우 Astra Control Provisioner가 설치되었는지 확인합니다 "활성화됨". Astra Control Provisioner는 23.10 이전 Astra Control Center 릴리즈에서 작동하지 않습니다. "Astra Control Provisioner를 업그레이드합니다" 최신 기능에 액세스하기 위해 업그레이드하는 Astra Control Center와 동일한 버전을 사용합니다.

3. Pod가 실행 중인지 확인합니다.

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

4. 스토리지 클래스가 지원되는 Astra Trident 드라이버를 사용하고 있는지 확인합니다. 공급자 이름은 이어야 합니다 `csi.trident.netapp.io`. 다음 예를 참조하십시오.

```
kubectl get sc
```

샘플 반응:

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

kubecononfig 파일을 생성합니다

kubecononfig 파일을 사용하여 Astra Control Service에 클러스터를 추가할 수 있습니다. 추가하려는 클러스터 유형에 따라 특정 단계를 사용하여 클러스터에 대한 kubeconfig 파일을 수동으로 생성해야 할 수 있습니다.

- Amazon EKS 클러스터용 kubbeconfig 파일을 생성합니다
- AWS(ROSA) 클러스터에서 Red Hat OpenShift Service에 대한 kubeconfig 파일을 생성합니다
- 다른 유형의 클러스터에 사용할 kubecononfig 파일을 생성합니다

Amazon EKS 클러스터용 kubbeconfig 파일을 생성합니다

다음 지침에 따라 아마존 EKS 클러스터에 대한 kubbeconfig 파일과 영구 토큰 암호를 생성하십시오. EKS에서 호스팅되는 클러스터에 영구 토큰 암호가 필요합니다.

단계

1. 아마존 문서의 지침에 따라 kubecononfig 파일을 생성합니다.

"Amazon EKS 클러스터에 대한 kubbeconfig 파일을 생성하거나 업데이트합니다"

2. 다음과 같이 서비스 계정을 생성합니다.

   a. 라는 서비스 계정 파일을 생성합니다 `astracontrol-service-account.yaml`.

      필요에 따라 서비스 계정 이름을 조정합니다. 네임스페이스 `kube-system` 은(는) 이러한 단계에 필요합니다. 여기서 서비스 계정 이름을 변경하는 경우 다음 단계에서 동일한 변경 사항을 적용해야 합니다.

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

+

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

3. 서비스 계정 적용:

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

4. 을 생성합니다 `ClusterRoleBinding` 파일을 호출했습니다 `astracontrol-clusterrolebinding.yaml`.

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

```
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. 클러스터 역할 바인딩을 적용합니다.

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

6. 라는 서비스 계정 토큰 암호 파일을 생성합니다 `astracontrol-secret.yaml`.

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

```
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. 토큰 암호 적용:

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

8. 토큰 암호 검색:

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

9. 를 교체합니다 `user` 다음 예제에서와 같이 토큰이 있는 AWS EKS kubecononfig 파일의 섹션입니다.

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

AWS(ROSA) 클러스터에서 Red Hat OpenShift Service에 대한 kubeconfig 파일을 생성합니다

다음 지침에 따라 AWS(ROSA) 클러스터에서 Red Hat OpenShift Service에 대한 kubeconfig 파일을 생성합니다.

단계

1. Rosa 클러스터에 로그인합니다.
2. 서비스 계정 생성:

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

3. 클러스터 역할 추가:

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

4. 다음 예제를 사용하여 서비스 계정 암호 구성 파일을 만듭니다.

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

```
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. 비밀 만들기:

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

6. 생성한 서비스 계정을 편집하고 Astra Control 서비스 계정 암호 이름을 에 추가합니다 `secrets` 섹션:

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

```
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. 교체 서비스 계정 암호를 나열합니다 `<CONTEXT>` 올바른 설치 상황:

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

출력의 끝은 다음과 유사합니다.

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

의 각 요소에 대한 인덱스입니다 `secrets` 어레이는 0으로 시작합니다. 위의 예에서 의 인덱스입니다 `astracontrol-service-account-dockercfg-dvfcd` 는 0이고 의 인덱스입니다 `secret-astracontrol-service-account` 1입니다. 출력에서 서비스 계정의 인덱스 번호를 기록해 둡니다. 다음 단계에서는 이 인덱스 번호가 필요합니다.

8. 다음과 같이 kubecononfig를 생성합니다.

   a. 을 생성합니다 `create-kubeconfig.sh` 파일. 대치 `TOKEN_INDEX` 다음 스크립트의 시작 부분에 올바른 값이 있습니다.

```
<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. Kubernetes 클러스터에 적용할 명령을 소스 하십시오.

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

9. (선택 사항) kubeconfig의 이름을 클러스터의 의미 있는 이름으로 바꿉니다.

```
mv kubeconfig-sa YOUR_CLUSTER_NAME_kubeconfig
```

다른 유형의 클러스터에 사용할 **kubecononfig** 파일을 생성합니다

다음 지침에 따라 Rancher, Upstream Kubernetes 및 Red Hat OpenShift 클러스터에 대한 제한적이거나 확장된 역할 kubeconfig 파일을 생성합니다.

kubeconfig를 사용하여 관리되는 클러스터의 경우 Astra Control Service에 대해 제한된 권한 또는 확장된 권한 관리자 역할을 선택적으로 생성할 수 있습니다.

다음 시나리오 중 하나가 사용자 환경에 적용되는 경우 이 절차를 통해 별도의 kubecononfig를 생성할 수 있습니다.

- 관리하는 클러스터에 대한 Astra Control 권한을 제한하려고 합니다
- 여러 개의 컨텍스트를 사용하며 설치 중에 구성된 기본 Astra Control kubecononfig를 사용할 수 없거나, 단일 컨텍스트의 제한된 역할은 사용자 환경에서 작동하지 않습니다

시작하기 전에

절차 단계를 완료하기 전에 관리하려는 클러스터에 대해 다음 사항을 확인해야 합니다.

- A "지원되는 버전입니다" kubectl이 설치되어 있습니다.
- Astra Control Service를 사용하여 추가하고 관리하려는 클러스터에 대한 kubectl 액세스

> ⓘ 이 절차를 수행하려면 Astra Control Service를 실행하는 클러스터에 액세스할 필요가 없습니다.

- 활성 컨텍스트에 대한 클러스터 관리자 권한으로 관리하려는 클러스터에 대한 활성 kubecononfig입니다

단계

1. 서비스 계정 생성:

a. 라는 서비스 계정 파일을 생성합니다 `astracontrol-service-account.yaml`.

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

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

b. 서비스 계정 적용:

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

2. Astra Control에서 클러스터를 관리할 수 있는 충분한 권한을 가진 다음 클러스터 역할 중 하나를 생성합니다.

제한된 클러스터 역할

이 역할에는 Astra Control에서 관리할 클러스터를 관리하는 데 필요한 최소 권한이 포함되어 있습니다.

a. 을 생성합니다 `ClusterRole` 호출되는 파일(예: `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 클러스터에만 해당) 의 끝에 다음을 추가합니다 `astra-admin-account.yaml` 파일:

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

c. 클러스터 역할 적용:

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

클러스터 역할이 확장되었습니다

이 역할에는 Astra Control에서 관리할 클러스터에 대한 확장된 권한이 포함됩니다. 여러 컨텍스트를 사용하고 설치 중에 구성된 기본 Astra Control kubecononfig를 사용할 수 없거나 단일 컨텍스트의 제한된 역할을 사용할 수 없는 경우 이 역할을 사용할 수 있습니다.

> 다음 사항을 참조하십시오 `ClusterRole` 일반 Kubernetes의 예는 단계입니다. 사용자 환경에 대한 지침은 Kubernetes 배포 문서를 참조하십시오.

a. 을 생성합니다 `ClusterRole` 호출되는 파일(예: `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. 클러스터 역할 적용:

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

3. 클러스터 역할에 대한 클러스터 역할 바인딩을 서비스 계정에 생성합니다.

a. 을 생성합니다 `ClusterRoleBinding` 파일을 호출했습니다 `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. 클러스터 역할 바인딩을 적용합니다.

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

4. 토큰 암호 생성 및 적용:
   a. 라는 토큰 비밀 파일을 만듭니다 `secret-astracontrol-service-account.yaml`.

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

```
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. 토큰 암호 적용:

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

5. 토큰 암호를 에 추가하여 서비스 계정에 추가합니다 `secrets` 배열(다음 예제의 마지막 줄):

```
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. 교체 서비스 계정 암호를 나열합니다 `<context>` 올바른 설치 상황:

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

출력의 끝은 다음과 유사합니다.

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

의 각 요소에 대한 인덱스입니다 `secrets` 어레이는 0으로 시작합니다. 위의 예에서 의 인덱스입니다 `astracontrol-service-account-dockercfg-48xhx` 는 0이고 의 인덱스입니다 `secret-astracontrol-service-account` 1입니다. 출력에서 서비스 계정의 인덱스 번호를 기록해 둡니다. 다음 단계에서는 이 인덱스 번호가 필요합니다.

7. 다음과 같이 kubecononfig를 생성합니다.
   a. 을 생성합니다 `create-kubeconfig.sh` 파일.
   b. 대치 `TOKEN_INDEX` 다음 스크립트의 시작 부분에 올바른 값이 있습니다.

```
<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. Kubernetes 클러스터에 적용할 명령을 소스 하십시오.

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

8. (선택 사항) kubeconfig의 이름을 클러스터의 의미 있는 이름으로 바꿉니다.

```
mv kubeconfig-sa YOUR_CLUSTER_NAME_kubeconfig
```