



클러스터를 추가합니다 Astra Control Service

NetApp
April 24, 2024

목차

Astra Control Service에 클러스터를 추가합니다.....	1
Astra Connector를 설치하여 클러스터를 관리한다	1
공급자 관리 클러스터를 추가합니다.....	7
자가 관리 클러스터를 추가합니다	16

Astra Control Service에 클러스터를 추가합니다

환경을 설정한 후에는 Kubernetes 클러스터를 생성하고 Astra Control Service에 추가할 준비가 된 것입니다. 이를 통해 Astra Control Service를 사용하여 클러스터의 애플리케이션을 보호할 수 있습니다.

Astra Control Service에 추가해야 하는 클러스터의 유형에 따라 클러스터를 추가하려면 다른 단계를 사용해야 합니다.

- **"공용 공급자 관리 클러스터를 Astra Control Service에 추가합니다"**: 다음 단계를 수행하여 공용 IP 주소가 있고 클라우드 공급자가 관리하는 클러스터를 추가할 수 있습니다. 클라우드 공급자의 서비스 주 계정, 서비스 계정 또는 사용자 계정이 필요합니다.
- **"Astra Control Service에 프라이빗 공급자 관리 클러스터를 추가합니다"**: 다음 단계를 수행하여 프라이빗 IP 주소가 있고 클라우드 공급자가 관리하는 클러스터를 추가할 수 있습니다. 클라우드 공급자의 서비스 주 계정, 서비스 계정 또는 사용자 계정이 필요합니다.
- **"공용 자가 관리 클러스터를 Astra Control Service에 추가합니다"**: 다음 단계를 사용하여 공용 IP 주소가 있고 조직에서 관리하는 클러스터를 추가할 수 있습니다. 추가하려는 클러스터에 대해 kubecononfig 파일을 생성해야 합니다.
- **"Astra Control Service에 프라이빗 자체 관리 클러스터를 추가합니다"**: 다음 단계를 사용하여 개인 IP 주소가 있고 조직에서 관리하는 클러스터를 추가할 수 있습니다. 추가하려는 클러스터에 대해 kubecononfig 파일을 생성해야 합니다.

Astra Connector를 설치하여 클러스터를 관리한다

Astra Connector는 관리형 클러스터에 상주하는 소프트웨어로, 관리형 클러스터와 Astra Control 간의 통신을 지원합니다. Astra Control Service를 사용하여 관리되는 클러스터의 경우 Astra Connector에 사용할 수 있는 2가지 버전이 있습니다.

- * 이전 버전의 Astra Connector *: **"이전 버전의 Astra Connector를 설치합니다"** Kubernetes 네이티브가 아닌 워크플로우를 통해 클러스터를 관리하려는 경우, 클러스터에 위치한다.
- [기술 미리보기] * 선언 Kubernetes Astra Connector *: **"선언적 Kubernetes 워크플로우로 관리되는 클러스터용 Astra Connector를 설치합니다"** 선언적 Kubernetes 워크플로우를 사용하여 클러스터를 관리하려는 경우 클러스터에 두어야 합니다. 클러스터에 Astra Connector를 설치하면 클러스터가 Astra Control에 자동으로 추가됩니다.



선언적 Kubernetes Astra Connector는 Astra Control EAP(Early Adopter Program)의 일부로만 제공됩니다. 에 가입하는 방법에 대한 자세한 내용은 NetApp 영업 담당자에게 문의하십시오.

이전 버전의 Astra Connector를 설치합니다

Astra Control Service는 이전 버전의 Astra Connector를 사용하여 Kubernetes가 아닌 네이티브 워크플로우로 관리되는 프라이빗 클러스터 와 Astra Control Service 간의 통신을 지원합니다. Kubernetes가 아닌 네이티브 워크플로우로 관리하려는 프라이빗 클러스터에 Astra Connector를 설치해야 합니다.

이전 버전의 Astra Connector는 Kubernetes가 아닌 네이티브 워크플로우로 관리되는 다음 유형의 프라이빗 클러스터를 지원합니다.

- Amazon Elastic Kubernetes Service(EKS)
- Azure Kubernetes 서비스(AKS)
- Google Kubernetes Engine(GKE)
- AWS 기반 Red Hat OpenShift Service(ROSA)
- AWS PrivateLink와 함께 Rosa
- Red Hat OpenShift Container Platform 온-프레미스

이 작업에 대해

- 이러한 단계를 수행할 때 Astra Control Service로 관리할 프라이빗 클러스터에 대해 다음 명령을 실행합니다.
- 방호 호스트를 사용하는 경우 방호 호스트의 명령줄에서 이러한 명령을 실행하십시오.

시작하기 전에

- Astra Control Service를 사용하여 관리할 프라이빗 클러스터에 액세스해야 합니다.
- 클러스터에 Astra Connector 연산자를 설치하려면 Kubernetes 관리자 권한이 필요합니다.

단계

1. Kubernetes가 아닌 네이티브 워크플로로 관리할 프라이빗 클러스터에 이전 Astra Connector 연산자를 설치합니다. 이 명령을 실행하면 네임스페이스가 생성됩니다 astra-connector-operator 이 만들어지고 구성이 네임스페이스에 적용됩니다.

```
kubectl apply -f https://github.com/NetApp/astra-connector-
operator/releases/download/23.07.0-
202310251519/astraconnector_operator.yaml
```

2. 작업자가 설치되어 있고 준비가 되었는지 확인합니다.

```
kubectl get all -n astra-connector-operator
```

3. Astra Control에서 API 토큰을 받습니다. 을 참조하십시오 "[Astra 자동화 문서](#)" 를 참조하십시오.
4. Astra-Connector 네임스페이스를 생성합니다.

```
kubectl create ns astra-connector
```

5. Astra Connector CR 파일을 생성하고 이름을 지정합니다 astra-connector-cr.yaml. 괄호 <> 의 값을 Astra Control 환경 및 클러스터 구성과 일치하도록 업데이트합니다.
 - * <ASTRA_CONTROL_SERVICE_URL> * : Astra Control Service의 웹 UI URL. 예를 들면 다음과 같습니다.

```
https://astra.netapp.io
```

- * <ASTRA_CONTROL_SERVICE_API_TOKEN> *: 이전 단계에서 얻은 Astra Control API 토큰.
- * <PRIVATE_AKS_CLUSTER_NAME> *: (AKS 클러스터만 해당) - 전용 Azure Kubernetes Service 클러스터의 클러스터 이름입니다. 전용 AKS 클러스터를 추가하는 경우에만 주석을 해제하고 이 줄을 채웁니다.
- * <ASTRA_CONTROL_ACCOUNT_ID> *: Astra Control 웹 UI에서 가져옵니다. 페이지 오른쪽 상단의 그림 아이콘을 선택하고 * API 액세스 * 를 선택합니다.

```
apiVersion: netapp.astraconnector.com/v1
kind: AstraConnector
metadata:
  name: astra-connector
  namespace: astra-connector
spec:
  natssync-client:
    cloud-bridge-url: <ASTRA_CONTROL_SERVICE_URL>
  imageRegistry:
    name: theotw
    secret: ""
  astra:
    token: <ASTRA_CONTROL_SERVICE_API_TOKEN>
    #clusterName: <PRIVATE_AKS_CLUSTER_NAME>
    accountId: <ASTRA_CONTROL_ACCOUNT_ID>
    acceptEULA: yes
```

6. 를 채운 후 astra-connector-cr.yaml 올바른 값이 있는 파일에 CR을 적용합니다.

```
kubectl apply -f astra-connector-cr.yaml
```

7. Astra Connector가 완전히 구축되었는지 확인:

```
kubectl get all -n astra-connector
```

8. 클러스터가 Astra Control에 등록되었는지 확인:

```
kubectl get astraconnector -n astra-connector
```

다음과 유사한 출력이 표시됩니다.

NAME	REGISTERED	ASTRACONNECTORID
STATUS		
astra-connector	true	be475ae5-1511-4eaa-9b9e-712f09b0d065
Registered with Astra		



ASTRACONNECTORID를 기록해 둡니다. Astra Control에 클러스터를 추가할 때 필요합니다.

다음 단계

Astra Connector를 설치했으므로 이제 프라이빗 클러스터를 Astra Control Service에 추가할 준비가 되었습니다.

- **"Astra Control Service에 프라이빗 공급자 관리 클러스터를 추가합니다"**: 다음 단계를 수행하여 프라이빗 IP 주소가 있고 클라우드 공급자가 관리하는 클러스터를 추가할 수 있습니다. 클라우드 공급자의 서비스 주 계정, 서비스 계정 또는 사용자 계정이 필요합니다.
- **"Astra Control Service에 프라이빗 자체 관리 클러스터를 추가합니다"**: 다음 단계를 사용하여 개인 IP 주소가 있고 조직에서 관리하는 클러스터를 추가할 수 있습니다. 추가하려는 클러스터에 대해 kubeconfig 파일을 생성해야 합니다.

를 참조하십시오

- **"클러스터를 추가합니다"**

(기술 미리 보기) 선언적 Kubernetes Astra Connector를 설치합니다

선언적 Kubernetes 워크플로우를 사용하여 관리되는 클러스터는 Astra Connector를 사용하여 관리형 클러스터와 Astra Control 간의 통신을 지원합니다. 선언적 Kubernetes 워크플로우로 관리할 모든 클러스터에 Astra Connector를 설치해야 합니다.

Kubernetes 명령 및 CR(Custom Resource) 파일을 사용하여 선언적 Kubernetes Astra Connector를 설치합니다.

이 작업에 대해

- 이러한 단계를 수행할 때 Astra Control으로 관리하려는 클러스터에서 다음 명령을 실행합니다.
- 방호 호스트를 사용하는 경우 방호 호스트의 명령줄에서 이러한 명령을 실행하십시오.

시작하기 전에

- Astra Control을 사용하여 관리할 클러스터에 액세스해야 합니다.
- 클러스터에 Astra Connector 연산자를 설치하려면 Kubernetes 관리자 권한이 필요합니다.



Kubernetes 1.25 이상 클러스터의 기본값인 Pod 보안 승인 적용을 사용하여 클러스터를 구성한 경우 해당 네임스페이스에 PSA 제한을 활성화해야 합니다. 을 참조하십시오 **"Astra Control을 사용하여 클러스터 관리를 위한 환경을 준비합니다"** 를 참조하십시오.

단계

1. 선언적 Kubernetes 워크플로우로 관리할 클러스터에 Astra Connector 연산자를 설치합니다. 이 명령을 실행하면 네임스페이스가 생성됩니다 astra-connector-operator 이 만들어지고 구성이 네임스페이스에 적용됩니다.

```
kubectl apply -f https://github.com/NetApp/astra-connector-  
operator/releases/download/24.02.0-  
202403151353/astraconnector_operator.yaml
```

2. 작업자가 설치되어 있고 준비가 되었는지 확인합니다.

```
kubectl get all -n astra-connector-operator
```

3. Astra Control에서 API 토큰을 받습니다. 을 참조하십시오 **"Astra 자동화 문서"** 를 참조하십시오.
4. 토큰을 사용하여 암호를 생성합니다. <API_TOKEN>를 Astra Control에서 받은 토큰으로 대체:

```
kubectl create secret generic astra-token \  
--from-literal=apiToken=<API_TOKEN> \  
-n astra-connector
```

5. Astra Connector 이미지를 가져오는 데 사용할 Docker 암호를 생성합니다. 괄호 안의 값을 사용자 환경의 정보로 대체:



<ASTRA_CONTROL_ACCOUNT_ID>는 Astra Control 웹 UI에서 찾을 수 있습니다. 웹 UI에서 페이지 오른쪽 상단의 그림 아이콘을 선택하고 * API access * 를 선택합니다.

```
kubectl create secret docker-registry regcred \  
--docker-username=<ASTRA_CONTROL_ACCOUNT_ID> \  
--docker-password=<API_TOKEN> \  
-n astra-connector \  
--docker-server=cr.astra.netapp.io
```

6. Astra Connector CR 파일을 생성하고 이름을 지정합니다 astra-connector-cr.yaml. 괄호 <> 의 값을 Astra Control 환경 및 클러스터 구성과 일치하도록 업데이트합니다.
- <ASTRA_CONTROL_ACCOUNT_ID>: 이전 단계에서 Astra Control 웹 UI에서 구함.
 - <CLUSTER_NAME>: Astra Control에서 이 클러스터를 할당해야 하는 이름입니다.
 - <ASTRA_CONTROL_URL>: Astra Control의 웹 UI URL입니다. 예를 들면 다음과 같습니다.

```
https://astra.control.url
```

```

apiVersion: astra.netapp.io/v1
kind: AstraConnector
metadata:
  name: astra-connector
  namespace: astra-connector
spec:
  astra:
    accountId: <ASTRA_CONTROL_ACCOUNT_ID>
    clusterName: <CLUSTER_NAME>
    #Only set `skipTLSValidation` to `true` when using the default
    self-signed
    #certificate in a proof-of-concept environment.
    skipTLSValidation: false #Should be set to false in production
    environments
    tokenRef: astra-token
  natsSyncClient:
    cloudBridgeURL: <ASTRA_CONTROL_HOST_URL>
  imageRegistry:
    name: cr.astra.netapp.io
    secret: regcred

```

7. 를 채운 후 astra-connector-cr.yaml 올바른 값이 있는 파일에 CR을 적용합니다.

```
kubectl apply -n astra-connector -f astra-connector-cr.yaml
```

8. Astra Connector가 완전히 구축되었는지 확인:

```
kubectl get all -n astra-connector
```

9. 클러스터가 Astra Control에 등록되었는지 확인:

```
kubectl get astraconnectors.astra.netapp.io -A
```

다음과 유사한 출력이 표시됩니다.

NAMESPACE	NAME	REGISTERED	ASTRACONNECTORID
STATUS			
astra-connector	astra-connector	true	00ac8-2cef-41ac-8777-
ed0583e	Registered with Astra		

10. Astra Control 웹 UI의 * 클러스터 * 페이지에서 관리되는 클러스터 목록에 클러스터가 나타나는지 확인합니다.

공급자 관리 클러스터를 추가합니다

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

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

- [Kubernetes 클러스터를 생성합니다](#)
- [클러스터를 Astra Control Service에 추가합니다](#)
- [기본 스토리지 클래스를 변경합니다](#)

Kubernetes 클러스터를 생성합니다

클러스터가 아직 없는 경우 요건에 맞는 클러스터를 생성할 수 있습니다 ["Amazon Elastic Kubernetes Service\(EKS\)에 대한 Astra Control 서비스 요구사항"](#). 클러스터가 아직 없는 경우 요건에 맞는 클러스터를 생성할 수 있습니다 ["Google Kubernetes Engine\(GKE\)에 대한 Astra Control Service 요구 사항"](#). 클러스터가 아직 없는 경우 요건에 맞는 클러스터를 생성할 수 있습니다 ["Azure NetApp Files를 사용하는 Azure Kubernetes Service\(AKS\)에 대한 Astra Control 서비스 요구사항"](#) 또는 ["Azure 관리 디스크를 사용하는 Azure Kubernetes Service\(AKS\)에 대한 Astra Control Service 요구 사항"](#).



Astra Control Service는 인증 및 ID 관리를 위해 Azure Active Directory(Azure AD)를 사용하는 AKS 클러스터를 지원합니다. 클러스터를 생성할 때 의 지침을 따릅니다 ["공식 문서"](#) Azure AD를 사용하도록 클러스터를 구성합니다. 클러스터가 AKS로 관리되는 Azure AD 통합에 대한 요구 사항을 충족하는지 확인해야 합니다.

클러스터를 Astra Control Service에 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.

Azure Kubernetes Service 및 Google Kubernetes Engine 클러스터를 관리할 때 Astra Control Provisioner 설치 및 라이프사이클 관리에 대한 두 가지 옵션이 있습니다.

- Astra Control Service를 사용하여 Astra Control Provisioner의 라이프사이클을 자동으로 관리할 수 있습니다. 이렇게 하려면 Astra Trident가 설치되어 있지 않고 Astra Control Service를 사용하여 관리하려는 클러스터에 Astra Control Provisioner가 활성화되어 있지 않아야 합니다. 이 경우 클러스터 관리를 시작할 때 Astra Control Service가 Astra Control Provisioner를 자동으로 활성화하며 Astra Control Provisioner 업그레이드는 자동으로 처리됩니다.
- Astra Control Provisioner의 라이프사이클을 직접 관리할 수 있습니다. 이렇게 하려면 Astra Control Service로 클러스터를 관리하기 전에 클러스터에서 Astra Control Provisioner를 활성화합니다. 이 경우 Astra Control Service는 Astra Control Provisioner가 이미 활성화되었음을 감지하여 Astra Control Provisioner 업그레이드를 다시 설치하거나 관리하지 않습니다. 을 참조하십시오 ["Astra Control Provisioner를 활성화합니다"](#) 단계를 위해 Astra Control Provisioner를 활성화합니다.

Astra Control Service를 사용하여 Amazon Web Services 클러스터를 관리할 때 Astra Control Provisioner에서만 사용할 수 있는 스토리지 백엔드가 필요한 경우 Astra Control Service를 통해 클러스터를 관리하기 전에 클러스터에서 Astra Control Provisioner를 수동으로 활성화해야 합니다. 을 참조하십시오 ["Astra Control Provisioner를 활성화합니다"](#) Astra Control Provisioner를 활성화하는 단계에 대해 알아보십시오.

Amazon Web Services에서 직접 지원합니다

- 클러스터를 생성한 IAM 사용자의 자격 증명이 포함된 JSON 파일이 있어야 합니다. ["IAM 사용자를 생성하는 방법을 알아봅니다"](#).
- Amazon FSx for NetApp ONTAP에는 Astra Control Provisioner가 필요합니다. Amazon FSx for NetApp ONTAP를 EKS 클러스터의 스토리지 백엔드로 사용할 계획인 경우 에서 Astra Control Provisioner 정보를 참조하십시오 ["EKS 클러스터 요구 사항"](#).
- (선택 사항) 제공해야 하는 경우 `kubectl` 클러스터 생성자가 아닌 다른 IAM 사용자에게 대한 클러스터에 대한 명령 액세스는 의 지침을 참조하십시오 ["Amazon EKS에서 클러스터를 생성한 후 다른 IAM 사용자 및 역할에 대한 액세스를 제공하려면 어떻게 해야 하나요?"](#).
- NetApp Cloud Volumes ONTAP를 스토리지 백엔드로 사용하려는 경우 Amazon Web Services와 연동되도록 Cloud Volumes ONTAP를 구성해야 합니다. Cloud Volumes ONTAP를 참조하십시오 ["설치 설명서"](#).

Microsoft Azure를 참조하십시오

- 서비스 보안 주체를 생성할 때 Azure CLI의 출력이 포함된 JSON 파일이 있어야 합니다. ["서비스 보안 주체를 설정하는 방법에 대해 알아봅니다"](#).

JSON 파일에 추가하지 않은 경우 Azure 구독 ID도 필요합니다.

- NetApp Cloud Volumes ONTAP를 스토리지 백엔드로 사용하려는 경우 Microsoft Azure와 연동하도록 Cloud Volumes ONTAP를 구성해야 합니다. Cloud Volumes ONTAP를 참조하십시오 ["설치 설명서"](#).

Google 클라우드

- 필요한 권한이 있는 서비스 계정에 대한 서비스 계정 키 파일이 있어야 합니다. ["서비스 계정 설정 방법에 대해 알아보십시오"](#).
- NetApp Cloud Volumes ONTAP를 스토리지 백엔드로 사용하려는 경우 Cloud Volumes ONTAP이 Google Cloud와 연동되도록 구성해야 합니다. Cloud Volumes ONTAP를 참조하십시오 ["설치 설명서"](#).

단계

1. (선택 사항) Amazon EKS 클러스터를 추가하거나 Astra Control Provisioner의 설치 및 업그레이드를 직접 관리하려는 경우 클러스터에서 Astra Control Provisioner를 활성화합니다. 을 참조하십시오 ["Astra Control Provisioner를 활성화합니다"](#) 참조하십시오.

2. 브라우저에서 Astra Control Service 웹 UI를 엽니다.

3. 대시보드에서 * Kubernetes 클러스터 관리 * 를 선택합니다.

표시되는 메시지에 따라 클러스터를 추가합니다.

4. * 공급자 *: 클라우드 공급자를 선택한 다음 새 클라우드 인스턴스를 생성하는 데 필요한 자격 증명을 제공하거나 사용할 기존 클라우드 인스턴스를 선택하십시오.
5. * Amazon Web Services *: JSON 파일을 업로드하거나 클립보드에서 해당 JSON 파일의 콘텐츠를 붙여넣어 Amazon Web Services IAM 사용자 계정에 대한 세부 정보를 제공합니다.

JSON 파일에는 클러스터를 생성한 IAM 사용자의 자격 증명이 포함되어야 합니다.

6. * Microsoft Azure *: JSON 파일을 업로드하거나 클립보드에서 해당 JSON 파일의 내용을 붙여넣어 Azure 서비스

보안 주체에 대한 세부 정보를 제공합니다.

JSON 파일에는 서비스 보안 주체를 생성할 때 Azure CLI의 출력이 포함되어야 합니다. 또한 구독 ID를 포함할 수 있으므로 Astra에 자동으로 추가됩니다. 그렇지 않으면 JSON을 제공한 후 ID를 수동으로 입력해야 합니다.

7. * Google Cloud Platform *: 파일을 업로드하거나 클립보드의 콘텐츠를 붙여 넣어 서비스 계정 키 파일을 제공합니다.

Astra Control Service는 서비스 계정을 사용하여 Google Kubernetes Engine에서 실행 중인 클러스터를 검색합니다.

8. * 기타 *: 이 탭은 자체 관리형 클러스터에만 사용됩니다.

a. * 클라우드 인스턴스 이름 *: 이 클러스터를 추가할 때 생성될 새 클라우드 인스턴스의 이름을 입력합니다. 에 대해 자세히 알아보십시오 ["클라우드 인스턴스"](#).

b. 다음 * 을 선택합니다.

Astra Control Service에는 선택할 수 있는 클러스터 목록이 표시됩니다.

c. * 클러스터 *: 목록에서 Astra Control Service에 추가할 클러스터를 선택합니다.



클러스터 목록에서 선택할 때는 * 자격 조건 * 열에 주의해야 합니다. 클러스터가 "부적격" 또는 "부분적으로 적격"인 경우 상태 위로 마우스를 가져가면 클러스터에 문제가 있는지 확인할 수 있습니다. 예를 들어 클러스터에 작업자 노드가 없는 것을 식별할 수 있습니다.

d. 다음 * 을 선택합니다.

e. (선택 사항) * 스토리지 *: 선택적으로 이 클러스터에 Kubernetes 애플리케이션을 배포할 스토리지 클래스를 선택하여 기본적으로 사용하도록 합니다.

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

10. 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"](#).

- 다음 * 을 선택합니다.
- * 검토 및 승인 *: 구성 세부 정보를 검토합니다.
- 클러스터를 Astra Control Service에 추가하려면 * 추가 * 를 선택합니다.

결과

이 클라우드 공급자를 위해 추가한 첫 번째 클러스터인 경우 Astra Control Service는 해당 클러스터에서 실행되는 애플리케이션 백업을 위해 클라우드 공급자용 오브젝트 저장소를 생성합니다. (이 클라우드 공급자에 후속 클러스터를 추가할 경우 더 이상 오브젝트 저장소가 생성되지 않습니다.) 기본 스토리지 클래스를 지정한 경우 Astra Control Service는 사용자가 지정한 기본 스토리지 클래스를 설정합니다. Amazon Web Services 또는 Google Cloud Platform에서 관리되는 클러스터의 경우 Astra Control Service는 클러스터에 관리자 계정도 생성합니다. 이 작업은 몇 분 정도 걸릴 수 있습니다.

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 공급자 관리 클러스터를 추가합니다

Astra Control Service를 사용하여 개인 GKE(Google Kubernetes Engine) 클러스터를 관리할 수 있습니다. 이 지침에서는 이미 전용 AKS 또는 OpenShift 클러스터를 만들고 원격으로 액세스할 수 있는 안전한 방법을 준비했다고 가정합니다. 전용 AKS 또는 OpenShift 클러스터를 만들고 액세스하는 방법에 대한 자세한 내용은 다음 문서를 참조하십시오.

- ["프라이빗 AKS 클러스터를 위한 Azure 설명서"](#)
- ["전용 OpenShift 클러스터를 위한 Azure 문서"](#)

Astra Control Service를 사용하여 AKS(프라이빗 Azure Kubernetes Service) 클러스터뿐 아니라 AKS의 프라이빗 Red Hat OpenShift 클러스터를 관리할 수 있습니다. 이 지침에서는 이미 전용 AKS 또는 OpenShift 클러스터를 만들고 원격으로 액세스할 수 있는 안전한 방법을 준비했다고 가정합니다. 전용 AKS 또는 OpenShift 클러스터를 만들고 액세스하는 방법에 대한 자세한 내용은 다음 문서를 참조하십시오.

- ["프라이빗 AKS 클러스터를 위한 Azure 설명서"](#)
- ["전용 OpenShift 클러스터를 위한 Azure 문서"](#)

Astra Control Service를 사용하여 전용 EKS(Amazon Elastic Kubernetes Service) 클러스터를 관리할 수 있습니다. 이 지침에서는 이미 전용 EKS 클러스터를 생성하고 원격으로 액세스할 수 있는 안전한 방법을 준비했다고 가정합니다. 전용 EKS 클러스터 생성 및 액세스에 대한 자세한 내용은 ["아마존 EKS 문서"](#)를 참조하십시오.

Astra Control Service에 프라이빗 클러스터를 추가하려면 다음 작업을 수행해야 합니다.

1. [Astra Connector](#)를 설치합니다
2. [영구 스토리지를 설정합니다](#)
3. [Astra Control Service에 프라이빗 공급자 관리 클러스터를 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"](#)

Astra Control Service에 프라이빗 공급자 관리 클러스터를 추가합니다

이제 Astra Control Service에 프라이빗 클러스터를 추가할 수 있습니다.

Azure Kubernetes Service 및 Google Kubernetes Engine 클러스터를 관리할 때 Astra Control Provisioner 설치 및 라이프사이클 관리에 대한 두 가지 옵션이 있습니다.

- Astra Control Service를 사용하여 Astra Control Provisioner의 라이프사이클을 자동으로 관리할 수 있습니다. 이렇게 하려면 Astra Trident가 설치되어 있지 않고 Astra Control Service를 사용하여 관리하려는 클러스터에 Astra Control Provisioner가 활성화되어 있지 않아야 합니다. 이 경우 클러스터 관리를 시작할 때 Astra Control Service가 Astra Control Provisioner를 자동으로 활성화하며 Astra Control Provisioner 업그레이드는 자동으로 처리됩니다.
- Astra Control Provisioner의 라이프사이클을 직접 관리할 수 있습니다. 이렇게 하려면 Astra Control Service로 클러스터를 관리하기 전에 클러스터에서 Astra Control Provisioner를 활성화합니다. 이 경우 Astra Control Service는 Astra Control Provisioner가 이미 활성화되었음을 감지하여 Astra Control Provisioner 업그레이드를 다시 설치하거나 관리하지 않습니다. 을 참조하십시오 ["Astra Control Provisioner를 활성화합니다"](#) 단계를 위해 Astra Control Provisioner를 활성화합니다.

Astra Control Service를 사용하여 Amazon Web Services 클러스터를 관리할 때 Astra Control Provisioner에서만 사용할 수 있는 스토리지 백엔드가 필요한 경우 Astra Control Service를 통해 클러스터를 관리하기 전에 클러스터에서 Astra Control Provisioner를 수동으로 활성화해야 합니다. 을 참조하십시오 ["Astra Control Provisioner를 활성화합니다"](#) Astra Control Provisioner를 활성화하는 단계에 대해 알아보십시오.

Amazon Web Services에서 직접 지원합니다

- 클러스터를 생성한 IAM 사용자의 자격 증명이 포함된 JSON 파일이 있어야 합니다. ["IAM 사용자를 생성하는 방법을 알아봅니다"](#).
- Amazon FSx for NetApp ONTAP에는 Astra Control Provisioner가 필요합니다. Amazon FSx for NetApp ONTAP를 EKS 클러스터의 스토리지 백엔드로 사용할 계획인 경우 Astra Control Provisioner 정보를 참조하십시오 ["EKS 클러스터 요구 사항"](#).
- (선택 사항) 제공해야 하는 경우 `kubectl` 클러스터 생성자가 아닌 다른 IAM 사용자에게 대한 클러스터에 대한 명령 액세스는 의 지침을 참조하십시오 ["Amazon EKS에서 클러스터를 생성한 후 다른 IAM 사용자 및 역할에 대한 액세스를 제공하려면 어떻게 해야 하나요?"](#).
- NetApp Cloud Volumes ONTAP를 스토리지 백엔드로 사용하려는 경우 Amazon Web Services와 연동되도록 Cloud Volumes ONTAP를 구성해야 합니다. Cloud Volumes ONTAP를 참조하십시오 ["설치 설명서"](#).

Microsoft Azure를 참조하십시오

- 서비스 보안 주체를 생성할 때 Azure CLI의 출력이 포함된 JSON 파일이 있어야 합니다. ["서비스 보안 주체를 설정하는 방법에 대해 알아봅니다"](#).

JSON 파일에 추가하지 않은 경우 Azure 구독 ID도 필요합니다.

- NetApp Cloud Volumes ONTAP를 스토리지 백엔드로 사용하려는 경우 Microsoft Azure와 연동하도록 Cloud Volumes ONTAP를 구성해야 합니다. Cloud Volumes ONTAP를 참조하십시오 ["설치 설명서"](#).

Google 클라우드

- 필요한 권한이 있는 서비스 계정에 대한 서비스 계정 키 파일이 있어야 합니다. ["서비스 계정 설정 방법에 대해 알아보십시오"](#).
- 클러스터가 프라이빗 인 경우, 를 참조하십시오 ["인증된 네트워크"](#) Astra Control Service IP 주소를 허용해야 합니다.

52.188.218.166/32

- NetApp Cloud Volumes ONTAP를 스토리지 백엔드로 사용하려는 경우 Cloud Volumes ONTAP이 Google Cloud와 연동되도록 구성해야 합니다. Cloud Volumes ONTAP를 참조하십시오 ["설치 설명서"](#).

단계

1. (선택 사항) Amazon EKS 클러스터를 추가하거나 Astra Control Provisioner의 설치 및 업그레이드를 직접 관리하려는 경우 클러스터에서 Astra Control Provisioner를 활성화합니다. 을 참조하십시오 ["Astra Control Provisioner를 활성화합니다"](#) 참조하십시오.

2. 브라우저에서 Astra Control Service 웹 UI를 엽니다.

3. 대시보드에서 * Kubernetes 클러스터 관리 * 를 선택합니다.

표시되는 메시지에 따라 클러스터를 추가합니다.

4. * 공급자 *: 클라우드 공급자를 선택한 다음 새 클라우드 인스턴스를 생성하는 데 필요한 자격 증명을 제공하거나 사용할 기존 클라우드 인스턴스를 선택하십시오.

5. * Amazon Web Services *: JSON 파일을 업로드하거나 클립보드에서 해당 JSON 파일의 콘텐츠를 붙여넣어

Amazon Web Services IAM 사용자 계정에 대한 세부 정보를 제공합니다.

JSON 파일에는 클러스터를 생성한 IAM 사용자의 자격 증명이 포함되어야 합니다.

6. * Microsoft Azure *: JSON 파일을 업로드하거나 클립보드에서 해당 JSON 파일의 내용을 붙여넣어 Azure 서비스 보안 주체에 대한 세부 정보를 제공합니다.

JSON 파일에는 서비스 보안 주체를 생성할 때 Azure CLI의 출력이 포함되어야 합니다. 또한 구독 ID를 포함할 수 있으므로 Astra에 자동으로 추가됩니다. 그렇지 않으면 JSON을 제공한 후 ID를 수동으로 입력해야 합니다.

7. * Google Cloud Platform *: 파일을 업로드하거나 클립보드의 콘텐츠를 붙여 넣어 서비스 계정 키 파일을 제공합니다.

Astra Control Service는 서비스 계정을 사용하여 Google Kubernetes Engine에서 실행 중인 클러스터를 검색합니다.

8. * 기타 *: 이 탭은 자체 관리형 클러스터에만 사용됩니다.

a. * 클라우드 인스턴스 이름 *: 이 클러스터를 추가할 때 생성될 새 클라우드 인스턴스의 이름을 입력합니다. 에 대해 자세히 알아보십시오 ["클라우드 인스턴스"](#).

b. 다음 * 을 선택합니다.

Astra Control Service에는 선택할 수 있는 클러스터 목록이 표시됩니다.

c. * 클러스터 *: 목록에서 Astra Control Service에 추가할 클러스터를 선택합니다.



클러스터 목록에서 선택할 때는 * 자격 조건 * 열에 주의해야 합니다. 클러스터가 "부적격" 또는 "부분적으로 적격"인 경우 상태 위로 마우스를 가져가면 클러스터에 문제가 있는지 확인할 수 있습니다. 예를 들어 클러스터에 작업자 노드가 없는 것을 식별할 수 있습니다.

9. 다음 * 을 선택합니다.

10. (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.

결과

이 클라우드 공급자를 위해 추가한 첫 번째 클러스터인 경우 Astra Control Service는 해당 클러스터에서 실행되는 애플리케이션 백업을 위해 클라우드 공급자용 오브젝트 저장소를 생성합니다. (이 클라우드 공급자에 후속 클러스터를 추가할 경우 더 이상 오브젝트 저장소가 생성되지 않습니다.) 기본 스토리지 클래스를 지정한 경우 Astra Control Service는 사용자가 지정한 기본 스토리지 클래스를 설정합니다. Amazon Web Services 또는 Google Cloud Platform에서 관리되는 클러스터의 경우 Astra Control Service는 클러스터에 관리자 계정도 생성합니다. 이 작업은 몇 분 정도 걸릴 수 있습니다.

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
```

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 kubeconfig 파일에 액세스할 수 있습니다. 를 따릅니다 ["참조하십시오"](#) kubeconfig 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 kubeconfig 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 프론트엔드, 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

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 Connector](#)를 설치합니다
2. [영구 스토리지를 설정](#)합니다

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 kubeconfig 파일에 액세스할 수 있습니다. 를 따릅니다 ["참조하십시오"](#) kubeconfig 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 kubeconfig 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 프론트엔드, 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

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"](#).

- 다음 * 을 선택합니다.
- * 검토 및 승인 *: 구성 세부 정보를 검토합니다.
- 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 `kubeconfig` 파일의 섹션입니다.

```
user:
  token: k8s-aws-
v1.aHR0cHM6Ly9zdHMudXMtd2VzdC0yLmFtYXpvc3cy5jb20vP0FjdGlvbj1HZXRDYWxsZ
XJJZGVudG10eSZWZXJzaW9uPTIwMTUyMDYtMTUwWC1BbXotQWxnb3JpdGhtPUFXUzQtSE1BQ
y1TSEEyNTYmWC1BbXotQ3JlZGVudG1hbD1BS01BM1JEWDDdKU0haWU9LSEQ2SyUyRjIwMjMwN
DAzJTJGdXMtd2VzdC0yJTJGc3RzJTJGYXdzNF9yZXF1ZXN0JlgtQW16LURhdGU9MjAyMzA0M
DNUMjA0MzQwWiZYLUFtei1FeHBpcmVzPTYwJlgtQW16LVNpZ25lZElhYWRlc3R5b250ZCUzQ
ngtazhzLWF3cy1pZCZYLUFtei1TaWduYXR1cmU9YjU4ZWZWM0NzdiM2NkZGYxNGRhNzU4MGI2Z
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=astraccontrol-service-account
NAMESPACE=default
NEW_CONTEXT=astraccontrol
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>
```

```
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>
```

```
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
- replicaset
- replicationcontrollers
- replicationcontrollers/scale
- rolebindings
- roles
- secrets
- serviceaccounts
- services
- statefulsets
- tridentmirrorrelationships
- tridentnapshotinfos
- 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 kubeconfig를 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>
```

```

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>
```

```

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 kubeconfig를 생성합니다.

- 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=astraccontrol-service-account
NAMESPACE=default
NEW_CONTEXT=astraccontrol
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
```

저작권 정보

Copyright © 2024 NetApp, Inc. All Rights Reserved. 미국에서 인쇄된 본 문서의 어떠한 부분도 저작권 소유자의 사전 서면 승인 없이는 어떠한 형식이나 수단(복사, 녹음, 녹화 또는 전자 검색 시스템에 저장하는 것을 비롯한 그래픽, 전자적 또는 기계적 방법)으로도 복제될 수 없습니다.

NetApp이 저작권을 가진 자료에 있는 소프트웨어에는 아래의 라이선스와 고지사항이 적용됩니다.

본 소프트웨어는 NetApp에 의해 '있는 그대로' 제공되며 상품성 및 특정 목적에의 적합성에 대한 명시적 또는 묵시적 보증을 포함하여(이에 제한되지 않음) 어떠한 보증도 하지 않습니다. NetApp은 대체품 또는 대체 서비스의 조달, 사용 불능, 데이터 손실, 이익 손실, 영업 중단을 포함하여(이에 국한되지 않음), 이 소프트웨어의 사용으로 인해 발생하는 모든 직접 및 간접 손해, 우발적 손해, 특별 손해, 징벌적 손해, 결과적 손해의 발생에 대하여 그 발생 이유, 책임론, 계약 여부, 엄격한 책임, 불법 행위(과실 또는 그렇지 않은 경우)와 관계없이 어떠한 책임도 지지 않으며, 이와 같은 손실의 발생 가능성이 통지되었다 하더라도 마찬가지입니다.

NetApp은 본 문서에 설명된 제품을 언제든지 예고 없이 변경할 권리를 보유합니다. NetApp은 NetApp의 명시적인 서면 동의를 받은 경우를 제외하고 본 문서에 설명된 제품을 사용하여 발생하는 어떠한 문제에도 책임을 지지 않습니다. 본 제품의 사용 또는 구매의 경우 NetApp에서는 어떠한 특허권, 상표권 또는 기타 지적 재산권이 적용되는 라이선스도 제공하지 않습니다.

본 설명서에 설명된 제품은 하나 이상의 미국 특허, 해외 특허 또는 출원 중인 특허로 보호됩니다.

제한적 권리 표시: 정부에 의한 사용, 복제 또는 공개에는 DFARS 252.227-7013(2014년 2월) 및 FAR 52.227-19(2007년 12월)의 기술 데이터-비상업적 품목에 대한 권리(Rights in Technical Data -Noncommercial Items) 조항의 하위 조항 (b)(3)에 설명된 제한사항이 적용됩니다.

여기에 포함된 데이터는 상업용 제품 및/또는 상업용 서비스(FAR 2.101에 정의)에 해당하며 NetApp, Inc.의 독점 자산입니다. 본 계약에 따라 제공되는 모든 NetApp 기술 데이터 및 컴퓨터 소프트웨어는 본질적으로 상업용이며 개인 비용만으로 개발되었습니다. 미국 정부는 데이터가 제공된 미국 계약과 관련하여 해당 계약을 지원하는 데에만 데이터에 대한 전 세계적으로 비독점적이고 양도할 수 없으며 재사용이 불가능하며 취소 불가능한 라이선스를 제한적으로 가집니다. 여기에 제공된 경우를 제외하고 NetApp, Inc.의 사전 서면 승인 없이는 이 데이터를 사용, 공개, 재생산, 수정, 수행 또는 표시할 수 없습니다. 미국 국방부에 대한 정부 라이선스는 DFARS 조항 252.227-7015(b)(2014년 2월)에 명시된 권한으로 제한됩니다.

상표 정보

NETAPP, NETAPP 로고 및 <http://www.netapp.com/TM>에 나열된 마크는 NetApp, Inc.의 상표입니다. 기타 회사 및 제품 이름은 해당 소유자의 상표일 수 있습니다.