Astra Control Center를 설정합니다

Astra Control Center

NetApp
April 25, 2024

목차

Astra Control Center를 설정합니다

Astra Control Center에 대한 라이센스를 추가합니다

Astra Control Center를 설치할 때 포함된 평가판 라이센스가 이미 설치되어 있습니다. Astra Control Center를 평가하는 경우 이 단계를 건너뛸 수 있습니다.

Astra Control UI 또는 를 사용하여 새 라이센스를 추가할 수 있습니다 "Astra Control API를 참조하십시오".

Astra Control Center 라이센스는 Kubernetes CPU 유닛을 사용하여 CPU 리소스를 측정하고, 모든 관리되는 Kubernetes 클러스터의 작업자 노드에 할당된 CPU 리소스를 고려합니다. 라이센스는 vCPU 사용량을 기준으로 합니다. 라이선스 계산 방법에 대한 자세한 내용은 을 참조하십시오 "라이센싱".

> 설치가 라이센스 CPU 유닛 수를 초과하여 증가할 경우, Astra Control Center를 통해 새 애플리케이션을 관리할 수 없습니다. 용량이 초과되면 경고가 표시됩니다.

> 기존 평가판 또는 전체 라이센스를 업데이트하려면 을 참조하십시오 "기존 라이센스를 업데이트합니다".

시작하기 전에

- 새로 설치된 Astra Control Center 인스턴스에 액세스합니다.
- 관리자 역할 권한.
- A "NetApp 라이센스 파일" (NLF)

단계

1. Astra Control Center UI에 로그인합니다.
2. 계정 * > * 라이센스 * 를 선택합니다.
3. 라이센스 추가 * 를 선택합니다.
4. 다운로드한 라이센스 파일(NLF)으로 이동합니다.
5. 라이센스 추가 * 를 선택합니다.

계정 * > * 라이센스 * 페이지에는 라이센스 정보, 만료 날짜, 라이센스 일련 번호, 계정 ID 및 사용된 CPU 단위가 표시됩니다.

> 평가판 라이센스가 있고 데이터를 AutoSupport로 전송하지 않는 경우, Astra Control Center에 장애가 발생할 경우 데이터 손실을 방지하기 위해 계정 ID를 저장해야 합니다.

Astra Control Provisioner를 활성화합니다

Astra Trident 버전 23.10 이상에는 라이센스를 보유한 Astra Control 사용자가 고급 스토리지 프로비저닝 기능에 액세스할 수 있도록 Astra Control Provisioner를 사용하는 옵션이 포함되어 있습니다. Astra Control Provisioner는 표준 Astra Trident CSI 기반 기능과 더불어 이 확장 기능을 제공합니다.

향후 Astra Control 업데이트에서 Astra Control Provisioner는 Astra Trident를 스토리지 프로비저닝 및 오케스트레이터로 대체하며 Astra Control을 사용하려면 필수입니다. 따라서 Astra Control 사용자가 Astra Control Provisioner를 활성화하는 것이 좋습니다. Astra Trident는 오픈 소스를 계속 유지하며, NetApp의 새로운 CSI 및 기타 기능으로 릴리즈, 유지, 지원 및 업데이트될 것입니다.

이 작업에 대해

Astra Control Center 사용이 허가된 사용자이고 Astra Control Provisioner 기능을 사용하려는 경우에는 이 절차를 따라야 합니다. Astra Trident 사용자가 Astra Control을 사용하지 않고 Astra Control Provisioner가 제공하는 추가 기능을 사용하려면 이 절차를 따라야 합니다.

각 사례에 대해 Astra Trident 24.02에서 Provisioner 기능이 기본적으로 활성화되어 있지 않으며 활성화되어야 합니다.

시작하기 전에

Astra Control Provisioner를 사용하도록 설정하려면 먼저 다음을 수행합니다.

Astra Control Center를 통해 사용자에게 **Astra Control Provisioner** 제공

- * Astra Control Center 라이센스 받기 *: 가 필요합니다 "Astra Control Center 라이센스" Astra Control Provisioner를 활성화하고 이 기능이 제공하는 기능에 액세스합니다.
- * Astra Control Center 23.10 이상 설치 또는 업그레이드 *: Astra Control과 함께 최신 Astra Control Provisioner 기능(24.02)을 사용하려면 최신 Astra Control Center 버전(24.02)이 필요합니다.
- * 클러스터에 AMD64 시스템 아키텍처가 있는지 확인 *: Astra Control Provisioner 이미지는 AMD64 및 ARM64 CPU 아키텍처 모두에서 제공되지만 Astra Control Center에서는 AMD64만 지원됩니다.
- * 레지스트리 액세스를 위한 Astra Control Service 계정 얻기 * : NetApp Support 사이트 대신 Astra Control 레지스트리를 사용하여 Astra Control Provisioner 이미지를 다운로드하려면 에 대한 등록을 완료하십시오 "Astra Control Service 계정". 양식을 작성하여 제출하고 BlueXP 계정을 생성하면 Astra Control Service 환영 이메일이 전송됩니다.
- * Astra Trident를 설치한 경우 해당 버전이 4개의 릴리즈 창 내에 있는지 확인 *: Astra Trident가 버전 24.02의 4개의 릴리즈 창 내에 있는 경우 Astra Control Provisioner를 사용하여 Astra Trident 24.02로 직접 업그레이드할 수 있습니다. 예를 들어, Astra Trident 23.04에서 24.02로 직접 업그레이드할 수 있습니다.

Astra Control Provisioner 전용 사용자

- * Astra Control Center 라이센스 받기 *: 가 필요합니다 "Astra Control Center 라이센스" Astra Control Provisioner를 활성화하고 이 기능이 제공하는 기능에 액세스합니다.
- * Astra Trident를 설치한 경우 해당 버전이 4개의 릴리즈 창 내에 있는지 확인 *: Astra Trident가 버전 24.02의 4개의 릴리즈 창 내에 있는 경우 Astra Control Provisioner를 사용하여 Astra Trident 24.02로 직접 업그레이드할 수 있습니다. 예를 들어, Astra Trident 23.04에서 24.02로 직접 업그레이드할 수 있습니다.
- * 레지스트리 액세스를 위한 Astra Control Service 계정 얻기 * : Astra Control Provisioner 이미지를 다운로드하려면 레지스트리에 액세스해야 합니다. 시작하려면 에 대한 등록을 완료하십시오 "Astra Control Service 계정". 양식을 작성하여 제출하고 BlueXP 계정을 생성하면 Astra Control Service 환영 이메일이 전송됩니다.

(1단계) **Astra Control Provisioner** 이미지를 가져옵니다

Astra Control Center 사용자는 Astra Control 레지스트리 또는 NetApp Support 사이트 방법을 사용하여 Astra Control Provisioner 이미지를 가져올 수 있습니다. Astra Control 없이 Astra Control Provisioner를 사용하려는 Astra Trident 사용자는 레지스트리 방법을 사용해야 합니다.

Astra Control 이미지 레지스트리

> 이 절차의 명령에 Docker 대신 Podman을 사용할 수 있습니다. Windows 환경을 사용하는 경우 PowerShell을 사용하는 것이 좋습니다.

1. NetApp Astra Control 이미지 레지스트리에 액세스:
   a. Astra Control Service 웹 UI에 로그온하여 페이지 오른쪽 상단의 그림 아이콘을 선택합니다.
   b. API 액세스 * 를 선택합니다.
   c. 계정 ID를 기록합니다.
   d. 같은 페이지에서 * API 토큰 생성 * 을 선택하고 API 토큰 문자열을 클립보드에 복사하여 편집기에 저장합니다.
   e. 원하는 방법을 사용하여 Astra Control 레지스트리에 로그인합니다.

```
docker login cr.astra.netapp.io -u <account-id> -p <api-token>
```

```
crane auth login cr.astra.netapp.io -u <account-id> -p <api-
token>
```

2. (사용자 지정 레지스트리에만 해당) 이미지를 사용자 지정 레지스트리로 이동하려면 다음 단계를 수행하십시오. 레지스트리를 사용하지 않는 경우 의 Trident 운영자 단계를 따르십시오 "다음 섹션을 참조하십시오".
   a. 레지스트리에서 Astra Control Provisioner 이미지를 가져옵니다.

> 가져온 이미지는 여러 플랫폼을 지원하지 않으며 Linux AMD64와 같이 이미지를 가져온 호스트와 동일한 플랫폼만 지원합니다.

```
docker pull cr.astra.netapp.io/astra/trident-acp:24.02.0
--platform <cluster platform>
```

예:

```
docker pull cr.astra.netapp.io/astra/trident-acp:24.02.0 --platform
linux/amd64
```

   a. 이미지에 태그 지정:

```
docker tag cr.astra.netapp.io/astra/trident-acp:24.02.0
<my_custom_registry>/trident-acp:24.02.0
```

b. 이미지를 사용자 지정 레지스트리에 푸시합니다.

```
docker push <my_custom_registry>/trident-acp:24.02.0
```

> 다음 Docker 명령을 실행하는 대신 크레인 복사본을 사용할 수 있습니다.
> `crane copy cr.astra.netapp.io/astra/trident-acp:24.02.0 <my_custom_registry>/trident-acp:24.02.0`

NetApp Support 사이트

1. Astra Control Provisioner 번들을 다운로드합니다 (`trident-acp-[version].tar`)를 선택합니다 "Astra Control Center 다운로드 페이지".
2. (권장 사항이지만 선택 사항) Astra Control Center(Astra-control-center-certs-[version].tar.gz)용 인증서 및 서명 번들을 다운로드하여 trident-acp-[version] tar 번들의 서명을 확인하십시오.

```
tar -vxzf astra-control-center-certs-[version].tar.gz
```

```
openssl dgst -sha256 -verify certs/AstraControlCenterDockerImages-public.pub -signature certs/trident-acp-[version].tar.sig trident-acp-[version].tar
```

3. Astra Control Provisioner 이미지 로드:

```
docker load < trident-acp-24.02.0.tar
```

응답:

```
Loaded image: trident-acp:24.02.0-linux-amd64
```

4. 이미지에 태그 지정:

```
docker tag trident-acp:24.02.0-linux-amd64 <my_custom_registry>/trident-acp:24.02.0
```

5. 이미지를 사용자 지정 레지스트리에 푸시합니다.

```
docker push <my_custom_registry>/trident-acp:24.02.0
```

(2단계) Astra Trident에서 Astra Control Provisioner를 사용하도록 설정합니다

원래 설치 방법으로 를 사용했는지 확인합니다 "연산자(수동 또는 Helm 사용) 또는 tridentctl" 그리고 원래 방법에 따라 적절한 단계를 완료합니다.

Astra Trident 운영자

1. "Astra Trident 설치 프로그램을 다운로드하여 압축을 풉니다".
2. Astra Trident를 아직 설치하지 않았거나 원본 Astra Trident 구축에서 연산자를 제거한 경우 다음 단계를 완료하십시오.
   a. CRD 생성:

   ```
   kubectl create -f
   deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.y
   aml
   ```

   b. 트라이덴트 네임스페이스를 만듭니다 (`kubectl create namespace trident`) 또는 트리덴트 네임스페이스가 여전히 존재하는지 확인합니다 (`kubectl get all -n trident`)를 클릭합니다. 네임스페이스가 제거된 경우 다시 만듭니다.
3. Astra Trident를 24.02.0으로 업데이트:

   > Kubernetes 1.24 이하 버전을 실행하는 클러스터의 경우, 를 사용합니다 `bundle_pre_1_25.yaml`. Kubernetes 1.25 이상을 실행하는 클러스터의 경우 를 사용합니다 `bundle_post_1_25.yaml`.

   ```
   kubectl -n trident apply -f trident-installer/deploy/<bundle-
   name.yaml>
   ```

4. Astra Trident가 실행 중인지 확인합니다.

   ```
   kubectl get torc -n trident
   ```

   응답:

   ```
   NAME       AGE
   trident    21m
   ```

5.] 비밀을 사용하는 레지스트리가 있는 경우 Astra Control Provisioner 이미지를 가져오는 데 사용할 비밀을 만듭니다.

   ```
   kubectl create secret docker-registry <secret_name> -n trident
   --docker-server=<my_custom_registry> --docker-username=<username>
   --docker-password=<token>
   ```

6. TridentOrchestrator CR을 편집하고 다음과 같이 편집합니다.

```
kubectl edit torc trident -n trident
```

a. Astra Trident 이미지에 대한 사용자 지정 레지스트리 위치를 설정하거나 Astra Control 레지스트리에서 가져옵니다 (`tridentImage: <my_custom_registry>/trident:24.02.0` 또는 `tridentImage: netapp/trident:24.02.0`)를 클릭합니다.

b. Astra Control Provisioner를 활성화합니다 (`enableACP: true`)를 클릭합니다.

c. Astra Control Provisioner 이미지의 사용자 지정 레지스트리 위치를 설정하거나 Astra Control 레지스트리에서 가져옵니다 (`acpImage: <my_custom_registry>/trident-acp:24.02.0` 또는 `acpImage: cr.astra.netapp.io/astra/trident-acp:24.02.0`)를 클릭합니다.

d. 를 설정했는지 확인합니다 이미지 풀 암호 이 절차의 앞부분에서 여기에서 설정할 수 있습니다 (`imagePullSecrets: - <secret_name>`)를 클릭합니다. 이전 단계에서 설정한 것과 동일한 이름 암호 이름을 사용합니다.

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  tridentImage: <registry>/trident:24.02.0
  enableACP: true
  acpImage: <registry>/trident-acp:24.02.0
  imagePullSecrets:
  - <secret_name>
```

7. 파일을 저장하고 종료합니다. 배포 프로세스가 자동으로 시작됩니다.

8. 운영자, 배포 및 복제 세트가 생성되었는지 확인합니다.

```
kubectl get all -n trident
```

> Kubernetes 클러스터에는 운영자의 인스턴스 * 하나가 있어야 합니다. Astra Trident 연산자를 여러 번 구축해서는 안 됩니다.

9. 를 확인합니다 `trident-acp` 컨테이너가 실행 중이며 `acpVersion` 있습니다 `24.02.0` 의 상태입니다 `Installed`:

```
kubectl get torc -o yaml
```

응답:

```
status:
  acpVersion: 24.02.0
  currentInstallationParams:
    ...
    acpImage: <registry>/trident-acp:24.02.0
    enableACP: "true"
    ...
  ...
  status: Installed
```

tridentctl 을 선택합니다

1. "Astra Trident 설치 프로그램을 다운로드하여 압축을 풉니다".
2. "기존 Astra Trident가 있는 경우 이를 호스팅하는 클러스터에서 제거합니다".
3. Astra Control Provisioner를 사용하도록 설정된 Astra Trident를 설치합니다 (`--enable-acp=true`):

```
./tridentctl -n trident install --enable-acp=true --acp
-image=mycustomregistry/trident-acp:24.02
```

4. Astra Control Provisioner가 활성화되었는지 확인합니다.

```
./tridentctl -n trident version
```

응답:

```
+----------------+----------------+-------------+ | SERVER VERSION |
CLIENT VERSION | ACP VERSION | +----------------+----------------
+-------------+ | 24.02.0 | 24.02.0 | 24.02.0. | +----------------
+----------------+-------------+
```

헬름

1. Astra Trident 23.07.1 이하를 설치한 경우 "설치 제거" 작업자 및 기타 구성품
2. Kubernetes 클러스터에서 1.24 이전 버전을 실행 중인 경우 psp:

```
kubectl delete psp tridentoperatorpod
```

3. Astra Trident Helm 리포지토리를 추가합니다.

```
helm repo add netapp-trident https://netapp.github.io/trident-helm-
chart
```

4. 제어 차트 업데이트:

```
helm repo update netapp-trident
```

응답:

```
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "netapp-trident" chart
repository
Update Complete. □Happy Helming!□
```

5. 영상을 나열합니다.

```
./tridentctl images -n trident
```

응답:

```
| v1.28.0            | netapp/trident:24.02.0|
|                    | docker.io/netapp/trident-autosupport:24.02|
|                    | registry.k8s.io/sig-storage/csi-
provisioner:v4.0.0|
|                    | registry.k8s.io/sig-storage/csi-
attacher:v4.5.0|
|                    | registry.k8s.io/sig-storage/csi-
resizer:v1.9.3|
|                    | registry.k8s.io/sig-storage/csi-
snapshotter:v6.3.3|
|                    | registry.k8s.io/sig-storage/csi-node-driver-
registrar:v2.10.0 |
|                    | netapp/trident-operator:24.02.0 (optional)
```

6. 트라이덴트 - 운전자 24.02.0을 사용할 수 있는지 확인합니다.

```
helm search repo netapp-trident/trident-operator --versions
```

응답:

```
NAME                            CHART VERSION   APP VERSION
DESCRIPTION
netapp-trident/trident-operator 100.2402.0      24.02.0         A
```

7. 사용 `helm install` 을 클릭하고 다음 설정을 포함하는 옵션 중 하나를 실행합니다.
   - 배포 위치의 이름입니다
   - Astra Trident 버전
   - Astra Control Provisioner 이미지의 이름
   - Provisioner를 활성화하는 플래그입니다
   - (선택 사항) 로컬 레지스트리 경로입니다. 로컬 레지스트리를 사용하는 경우, 을(를) 참조하십시오 "Trident 이미지" 하나의 레지스트리 또는 다른 레지스트리에 있을 수 있지만 모든 CSI 이미지는 동일한 레지스트리에 있어야 합니다.
   - Trident 네임스페이스

옵션

- 레지스트리가 없는 이미지

```
helm install trident netapp-trident/trident-operator --version
100.2402.0 --set acpImage=cr.astra.netapp.io/astra/trident-acp:24.02.0
--set enableACP=true --set operatorImage=netapp/trident-
operator:24.02.0 --set
tridentAutosupportImage=docker.io/netapp/trident-autosupport:24.02
--set tridentImage=netapp/trident:24.02.0 --namespace trident
```

- 하나 이상의 레지스트리에 있는 이미지

```
helm install trident netapp-trident/trident-operator --version
100.2402.0 --set acpImage=<your-registry>:<acp image> --set
enableACP=true --set imageRegistry=<your-registry>/sig-storage --set
operatorImage=netapp/trident-operator:24.02.0 --set
tridentAutosupportImage=docker.io/netapp/trident-autosupport:24.02
--set tridentImage=netapp/trident:24.02.0 --namespace trident
```

을 사용할 수 있습니다 `helm list` 이름, 네임스페이스, 차트, 상태, 앱 버전과 같은 설치 세부 정보를 검토하려면 수정본 번호.

Helm을 사용하여 Trident를 구축하는 데 문제가 있는 경우 다음 명령을 실행하여 Astra Trident를 완전히 제거합니다.

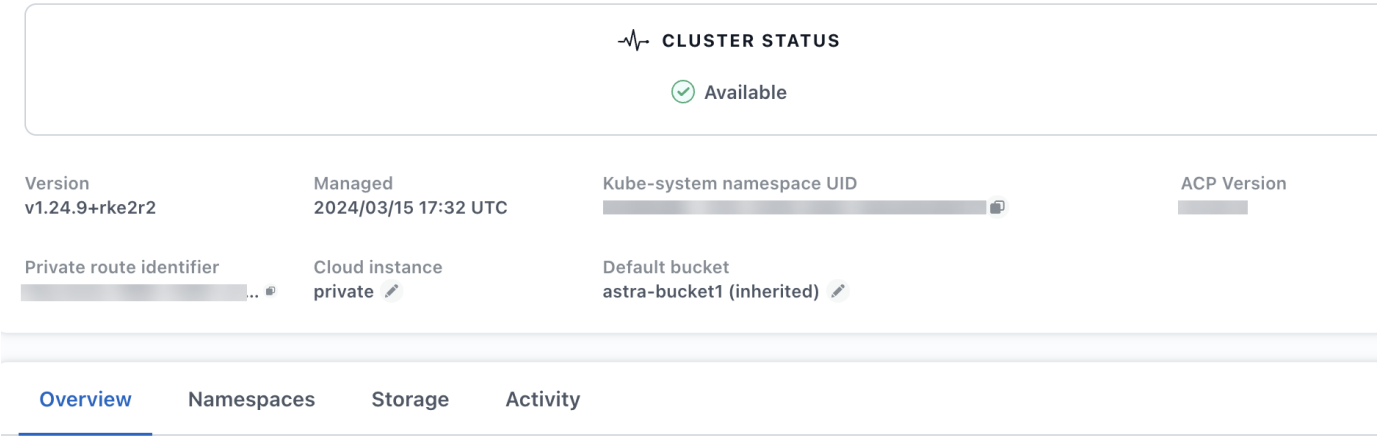
```
./tridentctl uninstall -n trident
```

- 하지 마십시오 * "Astra Trident CRD를 완전히 제거합니다" 설치 제거의 일부로 Astra Control Provisioner를 다시 활성화하려고 합니다.

결과

Astra Control Provisioner 기능이 활성화되어 있으며 실행 중인 버전에 제공되는 모든 기능을 사용할 수 있습니다.

(Astra Control Center 사용자만 해당) Astra Control Provisioner를 설치하면 Astra Control Center UI에서 Provisioner를 호스팅하는 클러스터에 가 표시됩니다 `ACP version` 을 사용하지 마십시오 `Trident version` 필드 및 현재 설치된 버전 번호



를 참조하십시오

- "Astra Trident 업그레이드 설명서"

Astra Control을 사용하여 클러스터 관리를 위한 환경을 준비합니다

클러스터를 추가하기 전에 다음 전제 조건이 충족되어야 합니다. 또한 자격 검사를 실행하여 클러스터를 Astra Control Center에 추가할 준비가 되었는지 확인하고 필요에 따라 kubeconfig 클러스터 역할을 생성해야 합니다.

Astra Control을 사용하면 환경 및 선호도에 따라 CR(Custom Resource) 또는 kubeconfig로 관리되는 클러스터를 추가할 수 있습니다.

시작하기 전에

- * 환경 필수 조건 충족 *: 사용자 환경이 충족됩니다 "구현할 수 있습니다" Astra Control Center의 경우
- * 작업자 노드 구성 *: 확인하십시오 "작업자 노드를 구성합니다" 클러스터에서 Pod가 백엔드 스토리지와 상호 작용할 수 있도록 적절한 스토리지 드라이버가 있어야 합니다.
- * PSA 제한 활성화 *: 클러스터에 Kubernetes 1.25 이상 클러스터의 표준인 Pod 보안 허용 적용이 활성화되어 있으면 이 네임스페이스에 PSA 제한을 활성화해야 합니다.
  - `netapp-acc-operator` 네임스페이스:

```
kubectl label --overwrite ns netapp-acc-operator pod-
security.kubernetes.io/enforce=privileged
```

  - `netapp monitoring` 네임스페이스:

```
kubectl label --overwrite ns netapp-monitoring pod-
security.kubernetes.io/enforce=privileged
```

- * ONTAP credentials *: Astra Control Center를 사용하여 앱을 백업 및 복원하려면 ONTAP 시스템에 ONTAP 자격 증명과 고급 사용자 및 사용자 ID가 설정되어 있어야 합니다.

  ONTAP 명령줄에서 다음 명령을 실행합니다.

```
export-policy rule modify -vserver <storage virtual machine name>
-policyname <policy name> -ruleindex 1 -superuser sys
export-policy rule modify -vserver <storage virtual machine name>
-policyname <policy name> -ruleindex 1 -anon 65534
```

- * kubeconfig-managed cluster requirements *: 이 요구 사항은 kubeconfig로 관리되는 앱 클러스터에 적용됩니다.
  - * kubeconfig 액세스 할 수 있습니다 * : 당신은 액세스 할 수 있습니다 "기본 클러스터 kubecononfig" 그것입니다 "설치하는 동안 를 구성했습니다".
  - * 인증 기관 고려 사항 *: 개인 CA(인증 기관)를 참조하는 kubeconfig 파일을 사용하여 클러스터를 추가하는 경우 에 다음 줄을 추가하십시오 `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:

자격 검사를 실행합니다

다음 자격 검사를 실행하여 클러스터를 Astra Control Center에 추가할 준비가 되었는지 확인합니다.

단계

1. 실행 중인 Astra Trident 버전을 확인합니다.

```
kubectl get tridentversion -n trident
```

Astra Trident가 있으면 다음과 유사한 출력이 표시됩니다.

```
NAME       VERSION
trident    24.02.0
```

Astra Trident가 없으면 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. 모든 Pod(포함)가 `trident-acp`) 실행 중:

```
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
```

클러스터 역할 kubecononfig를 생성합니다

kubeconfig를 사용하여 관리되는 클러스터의 경우 Astra Control Center에 대해 제한된 권한 또는 확장된 권한 관리자 역할을 선택적으로 생성할 수 있습니다. 이미 의 일부로 kubecononfig를 구성했으므로 Astra Control Center 설정에 필요한 절차는 아닙니다 "설치 프로세스".

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.

- kubbtl v1.23 이상이 설치되었습니다
- Astra Control Center를 통해 추가하고 관리하려는 클러스터에 kubctl 액세스를 허용합니다

> 이 절차를 수행하려면 Astra Control Center를 실행 중인 클러스터에 kubectl을 액세스할 필요가 없습니다.

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

단계

1. 서비스 계정 생성:
   a. astractrol-service-account.yaML이라는 서비스 계정 파일을 만듭니다.

```
<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. astracontrol-clusterrobinding.YAML이라는 ClusterRoleBinding 파일을 만듭니다.

```
<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. '<context>'을(를) 설치에 적합한 컨텍스트로 대체하여 서비스 계정 암호를 나열합니다.

```
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
```

(기술 미리 보기) 관리형 클러스터를 위한 **Astra Connector**를 설치합니다

Astra Control Center에서 관리되는 클러스터는 Astra Connector를 사용하여 관리형 클러스터와 Astra Control Center 간의 통신을 지원합니다. 관리하려는 모든 클러스터에 Astra Connector를 설치해야 합니다.

Astra Connector를 설치합니다

Kubernetes 명령 및 CR(Custom Resource) 파일을 사용하여 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. Astra Control으로 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
```

```yaml
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.

클러스터를 추가합니다

앱 관리를 시작하려면 Kubernetes 클러스터를 추가하고 이를 컴퓨팅 리소스로 관리합니다. Kubernetes 애플리케이션을 검색하려면 Astra Control Center용 클러스터를 추가해야 합니다.

> 관리를 위해 Astra Control Center에 다른 클러스터를 추가하기 전에 먼저 Astra Control Center에서 클러스터를 관리하는 것이 좋습니다. 메트릭 및 문제 해결을 위해 Kubemetrics 데이터 및 클러스터 관련 데이터를 전송하려면 관리 중인 초기 클러스터가 필요합니다.

시작하기 전에

- 클러스터를 추가하기 전에 필요한 를 검토 및 수행합니다 "선행 작업".
- ONTAP SAN 드라이버를 사용하는 경우 모든 Kubernetes 클러스터에서 다중 경로가 활성화되어 있는지 확인하십시오.

단계

1. 대시보드 또는 클러스터 메뉴에서 이동합니다.
   - 리소스 요약의 * 대시보드 * 에서 클러스터 창에서 * 추가 * 를 선택합니다.
   - 왼쪽 탐색 영역에서 * 클러스터 * 를 선택한 다음 클러스터 페이지에서 * 클러스터 추가 * 를 선택합니다.
2. Add Cluster * (클러스터 추가 *) 창이 열리면 kubecononfig.YAML 파일을 업로드하거나 kubecononfig.YAML 파일의 내용을 붙여 넣습니다.

   > "kubecononfig.yAML" 파일에는 하나의 클러스터에 대한 클러스터 자격 증명만 * 포함되어야 합니다.

   > 직접 만드는 경우 `kubeconfig` 파일에서 * 하나의 * 컨텍스트 요소만 정의해야 합니다. 을 참조하십시오 "Kubernetes 문서" 을 참조하십시오 `kubeconfig` 파일. 을 사용하여 제한된 클러스터 역할에 대해 kubecon무화과를 생성한 경우 "알려 드립니다"이 단계에서는 과베토화과를 업로드하거나 붙여 넣으십시오.

3. 자격 증명 이름을 제공하십시오. 기본적으로 자격 증명 이름은 클러스터 이름으로 자동 채워집니다.
4. 다음 * 을 선택합니다.
5. 이 Kubernetes 클러스터에 사용할 기본 스토리지 클래스를 선택하고 * Next * 를 선택합니다.

   > Astra Control Provisioner에서 구성되고 ONTAP 스토리지에서 지원하는 스토리지 클래스를 선택해야 합니다.

6. 정보를 검토하고 모든 것이 정상적으로 나타나면 * 추가 * 를 선택합니다.

결과

클러스터가 * 검색 * 상태로 전환되고 * 정상 * 으로 변경됩니다. 이제 Astra Control Center로 클러스터를 관리하고 있습니다.

> Astra Control Center에서 관리할 클러스터를 추가한 후 모니터링 연산자를 구축하는 데 몇 분이 걸릴 수 있습니다. 그 전까지는 알림 아이콘이 빨간색으로 바뀌고 * 모니터링 에이전트 상태 확인 실패 * 이벤트를 기록합니다. Astra Control Center가 올바른 상태를 획득하면 문제가 해결되므로 이 문제를 무시할 수 있습니다. 몇 분 이내에 문제가 해결되지 않으면 클러스터로 이동하여 를 실행합니다 `oc get pods -n netapp-monitoring` 시작점으로 사용됩니다. 문제를 디버깅하려면 모니터링 운영자 로그를 확인해야 합니다.

ONTAP 스토리지 백엔드에서 인증을 설정합니다

Astra Control Center는 ONTAP 백엔드를 인증하는 두 가지 모드를 제공합니다.

- * 자격 증명 기반 인증 *: 필요한 권한이 있는 ONTAP 사용자의 사용자 이름 및 암호입니다. ONTAP 버전과의 호환성을 최대화하려면 admin 또는 vsadmin과 같이 미리 정의된 보안 로그인 역할을 사용해야 합니다.
- * 인증서 기반 인증 *: Astra Control Center는 백엔드에 설치된 인증서를 사용하여 ONTAP 클러스터와 통신할 수도 있습니다. 클라이언트 인증서, 키 및 신뢰할 수 있는 CA 인증서를 사용해야 합니다(권장).

나중에 기존 백엔드를 업데이트하여 한 가지 인증 유형에서 다른 방법으로 이동할 수 있습니다. 한 번에 하나의 인증 방법만 지원됩니다.

자격 증명 기반 인증을 사용합니다

Astra Control Center에는 클러스터 범위에 대한 자격 증명이 필요합니다 `admin` ONTAP 백엔드와 통신합니다. 과 같이 미리 정의된 표준 역할을 사용해야 합니다 `admin`. 이를 통해 향후 Astra Control Center 릴리스에서 사용할 기능 API를 노출할 수 있는 향후 ONTAP 릴리스와 향후 호환될 수 있습니다.

> 사용자 지정 보안 로그인 역할은 Astra Control Center에서 생성 및 사용할 수 있지만 권장되지 않습니다.

백엔드 정의의 예는 다음과 같습니다.

```
{
  "version": 1,
  "backendName": "ExampleBackend",
  "storageDriverName": "ontap-nas",
  "managementLIF": "10.0.0.1",
  "dataLIF": "10.0.0.2",
  "svm": "svm_nfs",
  "username": "admin",
  "password": "secret"
}
```

백엔드 정의만 자격 증명이 일반 텍스트로 저장되는 곳입니다. 백엔드의 생성 또는 업데이트는 자격 증명에 대한 지식이 필요한 유일한 단계입니다. 따라서 Kubernetes 또는 스토리지 관리자가 수행할 수 있는 관리자 전용 작업입니다.

인증서 기반 인증을 사용합니다

Astra Control Center는 인증서를 사용하여 신규 및 기존 ONTAP 백엔드와 통신할 수 있습니다. 백엔드 정의에 다음 정보를 입력해야 합니다.

- `clientCertificate`: 클라이언트 인증서.
- `clientPrivateKey`: 연결된 개인 키.
- `trustedCACertificate`: 신뢰할 수 있는 CA 인증서입니다. 신뢰할 수 있는 CA를 사용하는 경우 이 매개변수를 제공해야 합니다. 신뢰할 수 있는 CA가 사용되지 않으면 이 작업을 무시할 수 있습니다.

다음 유형의 인증서 중 하나를 사용할 수 있습니다.

- 자체 서명된 인증서
- 타사 인증서입니다

자체 서명된 인증서를 사용하여 인증을 활성화합니다

일반적인 워크플로에는 다음 단계가 포함됩니다.

단계

1. 클라이언트 인증서 및 키를 생성합니다. 생성 시 CN(일반 이름)을 ONTAP 사용자로 설정하여 인증하십시오.

```
openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout k8senv.key
-out k8senv.pem -subj "/C=US/ST=NC/L=RTP/O=NetApp/CN=<common-name>"
```

2. 유형의 클라이언트 인증서를 설치합니다 `client-ca` ONTAP 클러스터의 키입니다.

```
security certificate install -type client-ca -cert-name <certificate-
name> -vserver <vserver-name>
security ssl modify -vserver <vserver-name> -client-enabled true
```

3. ONTAP 보안 로그인 역할이 인증서 인증 방법을 지원하는지 확인합니다.

```
security login create -user-or-group-name vsadmin -application ontapi
-authentication-method cert -vserver <vserver-name>
security login create -user-or-group-name vsadmin -application http
-authentication-method cert -vserver <vserver-name>
```

4. 생성된 인증서를 사용하여 인증을 테스트합니다. ONTAP 관리 LIF> 및 <vserver name>를 관리 LIF IP 및 SVM 이름으로 바꿉니다. LIF의 서비스 정책이 으로 설정되어 있는지 확인해야 합니다 `default-data-management`.

```
curl -X POST -Lk https://<ONTAP-Management-
LIF>/servlets/netapp.servlets.admin.XMLrequest_filer --key k8senv.key
--cert ~/k8senv.pem -d '<?xml version="1.0" encoding="UTF-8"?><netapp
xmlns=http://www.netapp.com/filer/admin version="1.21" vfiler="<vserver-
name>"><vserver-get></vserver-get></netapp>
```

5. 이전 단계에서 얻은 값을 사용하여 Astra Control Center UI에 스토리지 백엔드를 추가합니다.

타사 인증서로 인증을 활성화합니다

타사 인증서가 있는 경우 다음 단계를 사용하여 인증서 기반 인증을 설정할 수 있습니다.

단계

1. 개인 키와 CSR을 생성합니다.

```
openssl req -new -newkey rsa:4096 -nodes -sha256 -subj "/" -outform pem
-out ontap_cert_request.csr -keyout ontap_cert_request.key -addext
"subjectAltName = DNS:<ONTAP_CLUSTER_FQDN_NAME>,IP:<ONTAP_MGMT_IP>”
```

2. CSR을 Windows CA(타사 CA)로 전달하고 서명된 인증서를 발급합니다.
3. 서명된 인증서를 다운로드하고 이름을 'ONTAP_signed_cert.crt’로 지정합니다.
4. Windows CA(타사 CA)에서 루트 인증서를 내보냅니다.
5. 이 파일의 이름을 지정합니다 `ca_root.crt`

   이제 다음 세 개의 파일이 있습니다.

   - * 개인 키 *: `ontap_signed_request.key` (이 키는 ONTAP의 서버 인증서에 해당하는 키입니다. 서버 인증서를 설치하는 동안 필요합니다.)
   - * 서명된 인증서 *: `ontap_signed_cert.crt` (ONTAP에서 _server certificate_라고도 함)
   - * 루트 CA 인증서 *: `ca_root.crt` (ONTAP에서 _server-ca certificate_라고도 합니다.)
6. 이러한 인증서를 ONTAP에 설치합니다. 생성 및 설치 `server` 및 `server-ca` ONTAP의 인증서.

YAML의 샘플을 확장합니다

```
# Copy the contents of ca_root.crt and use it here.

security certificate install -type server-ca

Please enter Certificate: Press <Enter> when done

-----BEGIN CERTIFICATE-----
<certificate details>
-----END CERTIFICATE-----


You should keep a copy of the CA-signed digital certificate for
future reference.

The installed certificate's CA and serial number for reference:

CA:
serial:

The certificate's generated name for reference:


===

# Copy the contents of ontap_signed_cert.crt and use it here. For
key, use the contents of ontap_cert_request.key file.
security certificate install -type server
Please enter Certificate: Press <Enter> when done

-----BEGIN CERTIFICATE-----
<certificate details>
-----END CERTIFICATE-----

Please enter Private Key: Press <Enter> when done

-----BEGIN PRIVATE KEY-----
<private key details>
-----END PRIVATE KEY-----

Enter certificates of certification authorities (CA) which form the
certificate chain of the server certificate. This starts with the
issuing CA certificate of the server certificate and can range up to
the root CA certificate.
Do you want to continue entering root and/or intermediate
```

```
certificates {y|n}: n

The provided certificate does not have a common name in the subject
field.
Enter a valid common name to continue installation of the
certificate: <ONTAP_CLUSTER_FQDN_NAME>

You should keep a copy of the private key and the CA-signed digital
certificate for future reference.
The installed certificate's CA and serial number for reference:
CA:
serial:
The certificate's generated name for reference:


==
# Modify the vserver settings to enable SSL for the installed
certificate

ssl modify -vserver <vserver_name> -ca <CA>  -server-enabled true
-serial <serial number>       (security ssl modify)

==
# Verify if the certificate works fine:

openssl s_client -CAfile ca_root.crt -showcerts -servername server
-connect <ONTAP_CLUSTER_FQDN_NAME>:443
CONNECTED(00000005)
depth=1 DC = local, DC = umca, CN = <CA>
verify return:1
depth=0
verify return:1
write W BLOCK
---
Certificate chain
0 s:
   i:/DC=local/DC=umca/<CA>

-----BEGIN CERTIFICATE-----
<Certificate details>
```

7. 암호 없는 통신을 위해 동일한 호스트에 대한 클라이언트 인증서를 생성합니다. Astra Control Center는 이 프로세스를 사용하여 ONTAP와 통신합니다.
8. ONTAP에서 클라이언트 인증서 생성 및 설치:

YAML의 샘플을 확장합니다

```
# Use /CN=admin or use some other account which has privileges.
openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout
ontap_test_client.key -out ontap_test_client.pem -subj "/CN=admin"

Copy the content of ontap_test_client.pem file and use it in the
below command:
security certificate install -type client-ca -vserver <vserver_name>

Please enter Certificate: Press <Enter> when done

-----BEGIN CERTIFICATE-----
<Certificate details>
-----END CERTIFICATE-----

You should keep a copy of the CA-signed digital certificate for
future reference.
The installed certificate's CA and serial number for reference:

CA:
serial:
The certificate's generated name for reference:


==

ssl modify -vserver <vserver_name> -client-enabled true
(security ssl modify)

# Setting permissions for certificates
security login create -user-or-group-name admin -application ontapi
-authentication-method cert -role admin -vserver <vserver_name>

security login create -user-or-group-name admin -application http
-authentication-method cert -role admin -vserver <vserver_name>

==

#Verify passwordless communication works fine with the use of only
certificates:

curl --cacert ontap_signed_cert.crt  --key ontap_test_client.key
--cert ontap_test_client.pem
https://<ONTAP_CLUSTER_FQDN_NAME>/api/storage/aggregates
{
```

```
"records": [
{
"uuid": "f84e0a9b-e72f-4431-88c4-4bf5378b41bd",
"name": "<aggr_name>",
"node": {
"uuid": "7835876c-3484-11ed-97bb-d039ea50375c",
"name": "<node_name>",
"_links": {
"self": {
"href": "/api/cluster/nodes/7835876c-3484-11ed-97bb-d039ea50375c"
}
}
},
"_links": {
"self": {
"href": "/api/storage/aggregates/f84e0a9b-e72f-4431-88c4-
4bf5378b41bd"
}
}
}
],
"num_records": 1,
"_links": {
"self": {
"href": "/api/storage/aggregates"
}
}
}%
```


9. Astra Control Center UI에 스토리지 백엔드를 추가하고 다음 값을 제공합니다.
   - * 클라이언트 인증서 *: ONTAP_TEST_CLIENT.PEM
   - * 개인 키 *: ontap_test_client.key
   - * 신뢰할 수 있는 CA 인증서 *: ONTAP_signed_certt. CRT

스토리지 백엔드를 추가합니다

자격 증명 또는 인증서 인증 정보를 설정한 후 기존 ONTAP 스토리지 백엔드를 Astra Control Center에 추가하여 리소스를 관리할 수 있습니다.

Astra Control에서 스토리지 클러스터를 스토리지 백엔드로 관리하면 PVS(영구적 볼륨)와 스토리지 백엔드 간의 연결 및 추가 스토리지 메트릭을 얻을 수 있습니다.

Astra Control Provisioner를 사용하도록 설정한 경우 NetApp SnapMirror 기술을 사용할 때 Astra Control Center에서 ONTAP 스토리지 백엔드를 추가 및 관리하는 것은 선택 사항입니다.

단계

1. 왼쪽 탐색 영역의 대시보드에서 * backends * 를 선택합니다.
2. 추가 * 를 선택합니다.
3. 스토리지 백엔드 추가 페이지의 기존 사용 섹션에서 * ONTAP * 를 선택합니다.
4. 다음 중 하나를 선택합니다.
   - * 관리자 자격 증명 사용 *: ONTAP 클러스터 관리 IP 주소와 관리 자격 증명을 입력합니다. 자격 증명은 클러스터 전체의 자격 증명이어야 합니다.

     > 여기에 자격 증명을 입력한 사용자에게는 가 있어야 합니다 `ontapi` ONTAP 클러스터의 ONTAP System Manager에서 활성화된 사용자 로그인 액세스 방법입니다. SnapMirror 복제를 사용하려는 경우 액세스 방법이 있는 "admin" 역할의 사용자 자격 증명을 적용하십시오 `ontapi` 및 `http`, 소스 및 대상 ONTAP 클러스터 모두에서. 을 참조하십시오 "ONTAP 설명서에서 사용자 계정을 관리합니다" 를 참조하십시오.

   - * 인증서 사용 *: 인증서를 업로드합니다 `.pem` 파일, 인증서 키입니다 `.key` 파일 및 인증 기관 파일(옵션)을 선택합니다.
5. 다음 * 을 선택합니다.
6. 백엔드 세부 정보를 확인하고 * 관리 * 를 선택합니다.

결과

백엔드가 에 나타납니다 `online` 목록의 상태로 요약 정보를 표시합니다.

> 백엔드가 표시되도록 페이지를 새로 고쳐야 할 수 있습니다.

버킷을 추가합니다

Astra Control UI 또는 를 사용하여 버킷을 추가할 수 있습니다 "Astra Control API를 참조하십시오". 애플리케이션과 영구 스토리지를 백업하려는 경우나 클러스터 간에 애플리케이션을 클론 복제하려는 경우에는 오브젝트 저장소 버킷 공급자를 추가하는 것이 중요합니다. Astra Control은 이러한 백업 또는 클론을 정의한 오브젝트 저장소 버킷에 저장합니다.

애플리케이션 구성과 영구 스토리지를 동일한 클러스터에 클론 복제하려는 경우 Astra Control에 버킷이 필요하지 않습니다. 애플리케이션 스냅샷 기능에는 버킷이 필요하지 않습니다.

시작하기 전에

- Astra Control Center에서 관리하는 클러스터에서 연결할 수 있는 버킷이 있어야 합니다.
- 버킷에 대한 자격 증명이 있는지 확인하십시오.
- 버킷이 다음 유형 중 하나인지 확인합니다.
  - NetApp ONTAP S3
  - NetApp StorageGRID S3
  - Microsoft Azure를 참조하십시오

- 일반 S3

> AWS(Amazon Web Services) 및 GCP(Google Cloud Platform)는 일반 S3 버킷 유형을 사용합니다.

> Astra Control Center는 Amazon S3를 일반 S3 버킷 공급자로 지원하지만, Astra Control Center는 Amazon의 S3 지원을 주장하는 모든 오브젝트 저장소 공급업체를 지원하지 않을 수 있습니다.

단계

1. 왼쪽 탐색 영역에서 * Bucket * 을 선택합니다.
2. 추가 * 를 선택합니다.
3. 버킷 유형을 선택합니다.

> 버킷을 추가할 때 올바른 버킷 공급자를 선택하고 해당 공급자에 적합한 자격 증명을 제공합니다. 예를 들어, UI에서 NetApp ONTAP S3를 유형으로 받아들이고 StorageGRID 자격 증명을 받아들이지만, 이 버킷을 사용한 이후의 모든 애플리케이션 백업 및 복원이 실패합니다.

4. 기존 버킷 이름과 선택적 설명을 입력합니다.

> 버킷 이름과 설명은 나중에 백업을 생성할 때 선택할 수 있는 백업 위치로 나타납니다. 이 이름은 보호 정책 구성 중에도 표시됩니다.

5. S3 엔드포인트의 이름 또는 IP 주소를 입력합니다.
6. 자격 증명 선택 * 에서 * 추가 * 또는 * 기존 * 사용 탭을 선택합니다.
   - 추가 * 를 선택한 경우:
     i. Astra Control의 다른 자격 증명과 구별되는 자격 증명의 이름을 입력합니다.
     ii. 클립보드의 내용을 붙여 넣어 액세스 ID와 비밀 키를 입력합니다.
   - 기존 사용 * 을 선택한 경우:
     i. 버킷에 사용할 기존 자격 증명을 선택합니다.
7. 를 선택합니다 `Add`.

> 버킷을 추가하면 Astra Control이 기본 버킷 표시기로 하나의 버킷을 표시합니다. 사용자가 만든 첫 번째 버킷이 기본 버킷이 됩니다. 양동이 추가될 때 나중에 결정할 수 있습니다 "다른 기본 버킷을 설정합니다".