Astra Control Center를 설치합니다

Astra Control Center

NetApp
November 21, 2023

목차

표준 프로세스를 사용하여 **Astra Control Center**를 설치합니다

Astra Control Center를 설치하려면 NetApp Support 사이트에서 설치 번들을 다운로드하고 다음 단계를 수행하십시오. 이 절차를 사용하여 인터넷에 연결되었거나 공기가 연결된 환경에 Astra Control Center를 설치할 수 있습니다.

기타 설치 절차

- RedHat OpenShift OperatorHub * 로 설치: 이 옵션을 사용합니다 "대체 절차" OperatorHub를 사용하여 OpenShift에 Astra Control Center를 설치합니다.
- * Cloud Volumes ONTAP 백엔드를 사용하여 퍼블릭 클라우드에 설치 *: 사용 "수행할 수 있습니다" AWS(Amazon Web Services), GCP(Google Cloud Platform) 또는 Cloud Volumes ONTAP 스토리지 백엔드가 있는 Microsoft Azure에 Astra Control Center를 설치하려면 다음을 수행합니다.

Astra Control Center 설치 프로세스 데모는 를 참조하십시오 "이 비디오".

필요한 것

- "설치를 시작하기 전에 Astra Control Center 구축을 위한 환경을 준비합니다".
- 사용자 환경에서 POD 보안 정책을 구성했거나 구성하려는 경우 POD 보안 정책 및 해당 정책이 Astra Control Center 설치에 어떤 영향을 미치는지 숙지하십시오. 을 참조하십시오 "POD 보안 정책 제한 사항 이해".
- 모든 API 서비스가 정상 상태이며 사용 가능한지 확인합니다.

```
kubectl get apiservices
```

- 사용하려는 Astra FQDN이 이 클러스터에 라우팅될 수 있는지 확인합니다. 즉, 내부 DNS 서버에 DNS 항목이 있거나 이미 등록된 코어 URL 경로를 사용하고 있는 것입니다.
- 인증서 관리자가 클러스터에 이미 있는 경우 일부를 수행해야 합니다 "필수 단계" 따라서 Astra Control Center는 자체 인증 관리자를 설치하려고 시도하지 않습니다. 기본적으로 Astra Control Center는 설치 중에 자체 인증서 관리자를 설치합니다.

이 작업에 대해

Astra Control Center 설치 프로세스를 통해 다음을 수행할 수 있습니다.

- 에 Astra 구성 요소를 설치합니다 `netapp-acc` (또는 사용자 지정 이름) 네임스페이스입니다.
- 기본 Astra Control Owner 관리자 계정을 생성합니다.
- 관리자 이메일 주소와 기본 초기 설정 암호를 설정합니다. 이 사용자에게는 UI에 처음 로그인하는 데 필요한 소유자 역할이 할당됩니다.
- 모든 Astra Control Center Pod가 실행되고 있는지 확인합니다.
- Astra Control Center UI를 설치합니다.

Astra Control Center 운영자를 삭제하지 마십시오(예: `kubectl delete -f astra_control_center_operator_deploy.yaml`) 포드가 삭제되지 않도록 Astra Control Center 설치 또는 작동 중에 언제든지.

단계

Astra Control Center를 설치하려면 다음 단계를 수행하십시오.

- Astra Control Center를 다운로드하고 압축을 풉니다
- NetApp Astra kubtl 플러그인을 설치합니다
- 이미지를 로컬 레지스트리에 추가합니다
- 인증 요구 사항이 있는 레지스트리에 대한 네임스페이스 및 암호를 설정합니다
- Astra Control Center 운영자를 설치합니다
- Astra Control Center를 구성합니다
- Astra 제어 센터 및 운전자 설치를 완료합니다
- 시스템 상태를 확인합니다
- 부하 분산을 위한 수신 설정
- Astra Control Center UI에 로그인합니다

Astra Control Center를 다운로드하고 압축을 풉니다

1. 로 이동합니다 "Astra Control Center 평가판 다운로드 페이지" 를 방문하십시오.
2. Astra Control Center가 포함된 번들을 다운로드합니다 (`astra-control-center-[version].tar.gz`)를 클릭합니다.
3. (권장되지만 선택 사항) Astra Control Center용 인증서 및 서명 번들을 다운로드합니다 (`astra-control-center-certs-[version].tar.gz`)를 클릭하여 번들 서명을 확인합니다.

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

   ```
   openssl dgst -sha256 -verify certs/AstraControlCenter-public.pub
   -signature certs/astra-control-center-[version].tar.gz.sig astra-
   control-center-[version].tar.gz
   ```

   출력이 표시됩니다 `Verified OK` 확인 성공 후.

4. Astra Control Center 번들에서 이미지를 추출합니다.

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

NetApp Astra kubtl 플러그인을 설치합니다

NetApp Astra kubtl 명령줄 플러그인은 Astra Control Center 배포 및 업그레이드와 관련된 일반적인 작업을 수행할 때 시간을 절약해 줍니다.

필요한 것

NetApp은 다양한 CPU 아키텍처 및 운영 체제에 대한 플러그인 바이너리를 제공합니다. 이 작업을 수행하기 전에 사용 중인 CPU 및 운영 체제를 알아야 합니다.

단계

1. 사용 가능한 NetApp Astra kubectl 플러그인 바이너리를 나열하고 운영 체제 및 CPU 아키텍처에 필요한 파일 이름을 적어 주십시오.

   kubbeck 플러그인 라이브러리는 tar 번들의 일부이며 폴더에 압축이 풀립니다 `kubectl-astra`.

   ```
   ls kubectl-astra/
   ```

2. 올바른 바이너리를 현재 경로로 이동하고 이름을 로 변경합니다 `kubectl-astra`:

   ```
   cp kubectl-astra/<binary-name> /usr/local/bin/kubectl-astra
   ```

이미지를 로컬 레지스트리에 추가합니다

1. 용기 엔진에 적합한 단계 시퀀스를 완료합니다.

Docker 를 참조하십시오

1. 타볼의 루트 디렉토리로 변경합니다. 이 파일과 디렉토리가 표시됩니다.

   ```
   acc.manifest.bundle.yaml
   acc/
   ```

2. Astra Control Center 이미지 디렉토리의 패키지 이미지를 로컬 레지스트리에 밀어 넣습니다. 를 실행하기 전에 다음 대체 작업을 수행합니다 `push-images` 명령:

   - <BUNDLE_FILE>를 Astra Control 번들 파일의 이름으로 바꿉니다 (`acc.manifest.bundle.yaml`)를 클릭합니다.
   - <MY_FULL_REGISTRY_PATH>를 Docker 저장소의 URL로 바꿉니다. 예를 들어, "<a href="https://<docker-registry>"" class="bare">https://<docker-registry>".
   - <MY_REGISTRY_USER>를 사용자 이름으로 바꿉니다.
   - <MY_REGISTRY_TOKEN>를 레지스트리에 대한 인증된 토큰으로 바꿉니다.

     ```
     kubectl astra packages push-images -m <BUNDLE_FILE> -r
     <MY_FULL_REGISTRY_PATH> -u <MY_REGISTRY_USER> -p
     <MY_REGISTRY_TOKEN>
     ```

팟맨

1. 타볼의 루트 디렉토리로 변경합니다. 이 파일과 디렉토리가 표시됩니다.

   ```
   acc.manifest.bundle.yaml
   acc/
   ```

2. 레지스트리에 로그인합니다.

   ```
   podman login <YOUR_REGISTRY>
   ```

3. 사용하는 Podman 버전에 맞게 사용자 지정된 다음 스크립트 중 하나를 준비하고 실행합니다. <MY_FULL_REGISTRY_PATH>를 모든 하위 디렉토리가 포함된 리포지토리의 URL로 대체합니다.

   ```
   <strong>Podman 4</strong>
   ```

```
export REGISTRY=<MY_FULL_REGISTRY_PATH>
export PACKAGENAME=acc
export PACKAGEVERSION=22.11.0-82
export DIRECTORYNAME=acc
for astraImageFile in $(ls ${DIRECTORYNAME}/images/*.tar) ; do
astraImage=$(podman load --input ${astraImageFile} | sed 's/Loaded
image: //')
astraImageNoPath=$(echo ${astraImage} | sed 's:.*/::')
podman tag ${astraImageNoPath} ${REGISTRY}/netapp/astra/
${PACKAGENAME}/${PACKAGEVERSION}/${astraImageNoPath}
podman push ${REGISTRY}/netapp/astra/${PACKAGENAME}/${
PACKAGEVERSION}/${astraImageNoPath}
done
```

```
<strong>Podman 3</strong>
```

```
export REGISTRY=<MY_FULL_REGISTRY_PATH>
export PACKAGENAME=acc
export PACKAGEVERSION=22.11.0-82
export DIRECTORYNAME=acc
for astraImageFile in $(ls ${DIRECTORYNAME}/images/*.tar) ; do
astraImage=$(podman load --input ${astraImageFile} | sed 's/Loaded
image: //')
astraImageNoPath=$(echo ${astraImage} | sed 's:.*/::')
podman tag ${astraImageNoPath} ${REGISTRY}/netapp/astra/
${PACKAGENAME}/${PACKAGEVERSION}/${astraImageNoPath}
podman push ${REGISTRY}/netapp/astra/${PACKAGENAME}/${
PACKAGEVERSION}/${astraImageNoPath}
done
```

레지스트리 구성에 따라 스크립트가 만드는 이미지 경로는 다음과 같아야 합니다.
https://netappdownloads.jfrog.io/docker-astra-control-prod/netapp/astra/acc/22.11.0-82/image:version

인증 요구 사항이 있는 레지스트리에 대한 네임스페이스 및 암호를 설정합니다

1. Astra Control Center 호스트 클러스터에 대한 KUBECONFIG를 내보냅니다.

```
export KUBECONFIG=[file path]
```

설치를 완료하기 전에 KUBECONFIG가 Astra Control Center를 설치할 클러스터를 가리키고 있는지 확인하십시오. KUBECONFIG는 하나의 컨텍스트만 포함할 수 있습니다.

2. 인증이 필요한 레지스트리를 사용하는 경우 다음을 수행해야 합니다.

   a. 를 생성합니다 `netapp-acc-operator` 네임스페이스:

   ```
   kubectl create ns netapp-acc-operator
   ```

   응답:

   ```
   namespace/netapp-acc-operator created
   ```

   b. 에 대한 암호를 만듭니다 `netapp-acc-operator` 네임스페이스. Docker 정보를 추가하고 다음 명령을 실행합니다.

   자리 표시자입니다 `your_registry_path` 이전에 업로드한 이미지의 위치와 일치해야 합니다(예: `[Registry_URL]/netapp/astra/astracc/22.11.0-82`)를 클릭합니다.

   ```
   kubectl create secret docker-registry astra-registry-cred -n netapp-
   acc-operator --docker-server=[your_registry_path] --docker
   -username=[username] --docker-password=[token]
   ```

   샘플 반응:

   ```
   secret/astra-registry-cred created
   ```

   암호를 생성한 후 네임스페이스를 삭제하면 네임스페이스를 다시 만든 다음 네임스페이스에 대한 암호를 다시 생성합니다.

   c. 를 생성합니다 `netapp-acc` (또는 사용자 지정 이름) 네임스페이스입니다.

   ```
   kubectl create ns [netapp-acc or custom namespace]
   ```

   샘플 반응:

   ```
   namespace/netapp-acc created
   ```

d. 에 대한 암호를 만듭니다 `netapp-acc` (또는 사용자 지정 이름) 네임스페이스입니다. Docker 정보를 추가하고 다음 명령을 실행합니다.

```
kubectl create secret docker-registry astra-registry-cred -n [netapp-acc or custom namespace] --docker-server=[your_registry_path] --docker-username=[username] --docker-password=[token]
```

응답

```
secret/astra-registry-cred created
```

Astra Control Center 운영자를 설치합니다

1. 디렉토리를 변경합니다.

```
cd manifests
```

2. Astra Control Center 운영자 배포 YAML을 편집합니다 (`astra_control_center_operator_deploy.yaml`)를 클릭하여 로컬 레지스트리 및 암호를 참조합니다.

```
vim astra_control_center_operator_deploy.yaml
```

YAML 주석이 붙은 샘플은 다음 단계를 따릅니다.

a. 인증이 필요한 레지스트리를 사용하는 경우 의 기본 줄을 바꿉니다 `imagePullSecrets: []` 다음 포함:

```
imagePullSecrets:
- name: astra-registry-cred
```

b. 변경 `[your_registry_path]` 의 경우 `kube-rbac-proxy` 이미지를 에서 푸시한 레지스트리 경로로 이미지 이전 단계.

c. 변경 `[your_registry_path]` 의 경우 `acc-operator-controller-manager` 이미지를 에서 푸시한 레지스트리 경로로 이미지 이전 단계.

```
<strong>astra_control_center_operator_deploy.yaml</strong>
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  labels:
    control-plane: controller-manager
  name: acc-operator-controller-manager
  namespace: netapp-acc-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      control-plane: controller-manager
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        control-plane: controller-manager
    spec:
      containers:
      - args:
        - --secure-listen-address=0.0.0.0:8443
        - --upstream=http://127.0.0.1:8080/
        - --logtostderr=true
        - --v=10
        image: [your_registry_path]/kube-rbac-proxy:v4.8.0
        name: kube-rbac-proxy
        ports:
        - containerPort: 8443
          name: https
      - args:
        - --health-probe-bind-address=:8081
        - --metrics-bind-address=127.0.0.1:8080
        - --leader-elect
        env:
        - name: ACCOP_LOG_LEVEL
          value: "2"
        - name: ACCOP_HELM_INSTALLTIMEOUT
          value: 5m
        image: [your_registry_path]/acc-operator:[version x.y.z]
        imagePullPolicy: IfNotPresent
        livenessProbe:
          httpGet:
            path: /healthz
            port: 8081
          initialDelaySeconds: 15
          periodSeconds: 20
        name: manager
        readinessProbe:
```

```
          httpGet:
            path: /readyz
            port: 8081
          initialDelaySeconds: 5
          periodSeconds: 10
        resources:
          limits:
            cpu: 300m
            memory: 750Mi
          requests:
            cpu: 100m
            memory: 75Mi
        securityContext:
          allowPrivilegeEscalation: false
      imagePullSecrets: []
      securityContext:
        runAsUser: 65532
      terminationGracePeriodSeconds: 10
```

3. Astra Control Center 운영자를 설치합니다.

```
kubectl apply -f astra_control_center_operator_deploy.yaml
```

샘플 반응:

```
namespace/netapp-acc-operator created
customresourcedefinition.apiextensions.k8s.io/astracontrolcenters.astra.
netapp.io created
role.rbac.authorization.k8s.io/acc-operator-leader-election-role created
clusterrole.rbac.authorization.k8s.io/acc-operator-manager-role created
clusterrole.rbac.authorization.k8s.io/acc-operator-metrics-reader
created
clusterrole.rbac.authorization.k8s.io/acc-operator-proxy-role created
rolebinding.rbac.authorization.k8s.io/acc-operator-leader-election-
rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/acc-operator-manager-
rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/acc-operator-proxy-
rolebinding created
configmap/acc-operator-manager-config created
service/acc-operator-controller-manager-metrics-service created
deployment.apps/acc-operator-controller-manager created
```

4. Pod가 실행 중인지 확인합니다.

```
kubectl get pods -n netapp-acc-operator
```

Astra Control Center를 구성합니다

1. Astra Control Center 사용자 정의 리소스(CR) 파일을 편집합니다 (`astra_control_center.yaml`) 계정, 지원, 레지스트리 및 기타 필요한 구성을 만들려면:

```
vim astra_control_center.yaml
```

YAML 주석이 붙은 샘플은 다음 단계를 따릅니다.

2. 다음 설정을 수정하거나 확인합니다.

<code>accountName</code>

설정	지침	유형	예
`accountName`	를 변경합니다 `accountName` Astra Control Center 계정과 연결할 이름에 대한 문자열입니다. 하나의 accountName만 있을 수 있습니다.	문자열	`Example`

<code>astraVersion</code>

설정	지침	유형	예
`astraVersion`	배포할 Astra Control Center의 버전입니다. 값이 미리 채워질 수 있으므로 이 설정에 대한 작업은 필요하지 않습니다.	문자열	`22.11.0-82`

<code>astraAddress</code>

설정	지침	유형	예
`astraAddress`	를 변경합니다 `astraAddress` 브라우저에서 Astra Control Center에 액세스하기 위해 사용할 FQDN(권장) 또는 IP 주소에 대한 문자열입니다. 이 주소는 Astra Control Center가 데이터 센터에서 어떻게 검색되는지 정의하며, 이 주소를 완료하면 로드 밸런서에서 제공한 것과 동일한 FQDN 또는 IP 주소입니다 "Astra Control Center 요구 사항". 참고: 사용하지 마십시오 `http://` 또는 `https://` 를 입력합니다. 에서 사용하기 위해 이 FQDN을 복사합니다 나중에.	문자열	`astra.example.com`

<code>autoSupport</code>

이 섹션에서 어떤 항목을 선택하는지에 따라 NetApp의 사전 지원 애플리케이션인 NetApp Active IQ에 참여할 것인지, 그리고 데이터를 보낼 위치를 결정할 수 있습니다. 인터넷 연결이 필요하며(포트 442) 모든 지원 데이터가 익명화됩니다.

설정	사용	지침	유형	예
`autoSupport.enrolled`	둘 다 가능합니다 `enrolled` 또는 `url` 필드를 선택해야 합니다	변경 `enrolled` 을 눌러 AutoSupport to로 이동합니다 `false` 인터넷 연결이 없거나 보관되지 않은 사이트의 경우 `true` 연결된 사이트의 경우. 의 설정 `true` 지원을 위해 익명 데이터를 NetApp에 전송할 수 있습니다. 기본 선택 옵션은 입니다 `false` 및 은 NetApp에 지원 데이터가 전송되지 않음을 나타냅니다.	부울	`false` (이 값은 기본값입니다.)
`autoSupport.url`	둘 다 가능합니다 `enrolled` 또는 `url` 필드를 선택해야 합니다	이 URL은 익명 데이터를 보낼 위치를 결정합니다.	문자열	`https://support.netapp.com/asupprod/post/1.0/postAsup`

<code>email</code>

설정	지침	유형	예
`email`	를 변경합니다 `email` 문자열을 기본 초기 관리자 주소로 설정합니다. 에서 사용할 이 이메일 주소를 복사합니다 나중에. 이 이메일 주소는 UI에 로그인할 초기 계정의 사용자 이름으로 사용되며 Astra Control에서 이벤트를 알립니다.	문자열	`admin@example.com`

<code>firstName</code>

설정	지침	유형	예
`firstName`	Astra 계정과 연결된 기본 초기 관리자의 이름입니다. 여기에 사용된 이름은 처음 로그인한 후 UI의 제목에 표시됩니다.	문자열	`SRE`

<code>LastName</code>

설정	지침	유형	예
`lastName`	Astra 계정과 연결된 기본 초기 관리자의 성. 여기에 사용된 이름은 처음 로그인한 후 UI의 제목에 표시됩니다.	문자열	`Admin`

<code>imageRegistry</code>

이 섹션에서 선택한 사항은 Astra 응용 프로그램 이미지, Astra Control Center Operator 및 Astra Control Center Helm 리포지토리를 호스팅하는 컨테이너 이미지 레지스트리를 정의합니다.

설정	사용	지침	유형	예
`imageRegistry.name`	필수 요소입니다	에서 이미지를 푸시한 이미지 레지스트리의 이름입니다 이전 단계. 사용하지 마십시오 `http://` 또는 `https://` 레지스트리 이름.	문자열	`example.registry.com/astra`
`imageRegistry.secret`	에 대해 입력한 문자열인 경우 필수입니다 `imageRegistry.name' requires a secret.` `IMPORTANT: If you are using a registry that does not require authorization, you must delete this` `secret` 줄 내부 `imageRegistry` 그렇지 않으면 설치가 실패합니다.	이미지 레지스트리를 인증하는 데 사용되는 Kubernetes 비밀의 이름입니다.	문자열	`astra-registry-cred`

<code>storageClass</code>

설정	지침	유형	예
`storageClass`	를 변경합니다 `storageClass` 값 시작 `ontap-gold` 을 다른 Trident storageClass 리소스에 액세스하십시오. 명령을 실행합니다 `kubectl get sc` 구성된 기존 스토리지 클래스를 확인하려면 다음을 수행합니다. 매니페스트 파일에 Trident 기반 스토리지 클래스 중 하나를 입력해야 합니다 (`astra-control-center-<version>.manifest`) 및 는 Astra PVS에 사용됩니다. 이 옵션이 설정되어 있지 않으면 기본 스토리지 클래스가 사용됩니다. 참고: 기본 스토리지 클래스가 구성된 경우 기본 주석이 있는 유일한 스토리지 클래스인지 확인하십시오.	문자열	`ontap-gold`

<code>volumeReclaimPolicy</code>

설정	지침	유형	옵션
`volumeReclaimPolicy`	그러면 Astra의 PVS에 대한 재확보 정책이 설정됩니다. 이 정책을 으로 설정합니다 `Retain` Astra가 삭제된 후 영구 볼륨을 유지합니다. 이 정책을 으로 설정합니다 `Delete` Astra가 삭제된 후 영구 볼륨을 삭제합니다. 이 값을 설정하지 않으면 PVS가 유지됩니다.	문자열	• `Retain` (기본값) • `Delete`

<code>ingressType</code>

설정	지침	유형	옵션
`ingressType`	다음 수신 유형 중 하나를 사용하십시오.**`Generic`** (`ingressType: "Generic"`) (기본값) 다른 수신 컨트롤러를 사용 중이거나 자체 수신 컨트롤러를 사용하려는 경우 이 옵션을 사용합니다. Astra Control Center를 배포한 후 을 구성해야 합니다 "수신 컨트롤러" URL을 사용하여 Astra Control Center를 표시합니다 .**`AccTraefik`** (`ingressType: "AccTraefik"`) 수신 컨트롤러를 구성하지 않으려는 경우 이 옵션을 사용하십시오. 그러면 Astra Control Center가 구축됩니다 `traefik` Kubernetes 로드 밸런서 유형 서비스로서의 게이트웨이 Astra Control Center는 "loadbalancer" 유형의 서비스를 사용합니다. (`svc/traefik` Astra Control Center 네임스페이스에서), 액세스 가능한 외부 IP 주소를 할당해야 합니다. 로드 밸런서가 사용자 환경에서 허용되고 아직 로드 밸런서가 구성되어 있지 않은 경우 MetalLB 또는 다른 외부 서비스 로드 밸런서를 사용하여 외부 IP 주소를 서비스에 할당할 수 있습니다. 내부 DNS 서버 구성에서 Astra Control Center에 대해 선택한 DNS 이름을 부하 분산 IP 주소로 지정해야 합니다. 참고: "로드 밸런서" 및 수신 서비스 유형에 대한 자세한 내용은 을 참조하십시오 "요구 사항".	문자열	• `Generic` (기본값) • `AccTraefik`

<code>astraResourcesScaler</code>

설정	지침	유형	옵션
`astraResourcesScaler`	AstraControlCenter 리소스 제한에 대한 확장 옵션 기본적으로 Astra Control Center는 Astra 내의 대부분의 구성 요소에 대해 설정된 리소스 요청과 함께 배포됩니다. 이 구성을 통해 Astra Control Center 소프트웨어 스택은 애플리케이션 로드 및 확장 수준이 높은 환경에서 더 나은 성능을 발휘할 수 있습니다. 그러나 더 작은 개발 또는 테스트 클러스터를 사용하는 시나리오에서는 CR 필드를 사용합니다 `astraResourcesScalar` 로 설정할 수 있습니다 `Off`. 이렇게 하면 리소스 요청이 비활성화되고 소규모 클러스터에 구축할 수 있습니다.	문자열	• `Default` (기본값) • `Off`

<code>crds</code>

이 섹션에서 선택한 사항은 Astra Control Center에서 CRD를 처리하는 방법을 결정합니다.

설정	지침	유형	예
`crds.externalCertManager`	외부 인증서 관리자를 사용하는 경우 를 변경합니다 `externalCertManager` 를 선택합니다 `true`. 기본값입니다 `false` 설치 중에 Astra Control Center가 자체 인증서 관리자 CRD를 설치합니다. CRD는 클러스터 전체 오브젝트이며 이를 설치하면 클러스터의 다른 부분에 영향을 줄 수 있습니다. 이 플래그를 사용하여 Astra Control Center에 이러한 CRD가 Astra Control Center 외부의 클러스터 관리자에 의해 설치 및 관리된다는 신호를 보낼 수 있습니다.	부울	`False` (이 값은 기본값입니다.)
`crds.externalTraefik`	기본적으로 Astra Control Center는 필요한 Traefik CRD를 설치합니다. CRD는 클러스터 전체 오브젝트이며 이를 설치하면 클러스터의 다른 부분에 영향을 줄 수 있습니다. 이 플래그를 사용하여 Astra Control Center에 이러한 CRD가 Astra Control Center 외부의 클러스터 관리자에 의해 설치 및 관리된다는 신호를 보낼 수 있습니다.	부울	`False` (이 값은 기본값입니다.)

```
<strong>astra_control_center.yaml</strong>
```

```
apiVersion: astra.netapp.io/v1
kind: AstraControlCenter
metadata:
  name: astra
spec:
  accountName: "Example"
  astraVersion: "ASTRA_VERSION"
  astraAddress: "astra.example.com"
  autoSupport:
    enrolled: true
  email: "[admin@example.com]"
  firstName: "SRE"
  lastName: "Admin"
  imageRegistry:
    name: "[your_registry_path]"
    secret: "astra-registry-cred"
  storageClass: "ontap-gold"
  volumeReclaimPolicy: "Retain"
  ingressType: "Generic"
  astraResourcesScaler: "Default"
  additionalValues: {}
  crds:
    externalTraefik: false
    externalCertManager: false
```

Astra 제어 센터 및 운전자 설치를 완료합니다

1. 이전 단계에서 아직 작성하지 않은 경우 를 만듭니다 `netapp-acc` (또는 사용자 지정) 네임스페이스:

   ```
   kubectl create ns [netapp-acc or custom namespace]
   ```

   샘플 반응:

   ```
   namespace/netapp-acc created
   ```

2. 에 Astra Control Center를 설치합니다 `netapp-acc` (또는 사용자 지정) 네임스페이스:

   ```
   kubectl apply -f astra_control_center.yaml -n [netapp-acc or custom
   namespace]
   ```

   샘플 반응:

```
astracontrolcenter.astra.netapp.io/astra created
```

시스템 상태를 확인합니다

kubeck 명령을 사용하여 시스템 상태를 확인할 수 있습니다. OpenShift를 사용하려는 경우 검증 단계에 유사한 OC 명령을 사용할 수 있습니다.

단계

1. 모든 시스템 구성 요소가 성공적으로 설치되었는지 확인합니다.

```
kubectl get pods -n [netapp-acc or custom namespace]
```

   각 POD의 상태는 입니다 `Running`. 시스템 포드를 구축하는 데 몇 분 정도 걸릴 수 있습니다.

샘플 응답

```
NAME                                          READY   STATUS
RESTARTS        AGE
acc-helm-repo-76d8d845c9-ggds2                1/1     Running   0
14m
activity-6cc67ff9f4-z48mr                     1/1     Running   2
(8m32s ago)   9m
api-token-authentication-7s67v                1/1     Running   0
8m56s
api-token-authentication-bplb4                1/1     Running   0
8m56s
api-token-authentication-p2c9z                1/1     Running   0
8m56s
asup-6cdfbc6795-md8vn                         1/1     Running   0
9m14s
authentication-9477567db-8hnc9                1/1     Running   0
7m4s
bucketservice-f4dbdfcd6-wqzkw                 1/1     Running   0
8m48s
cert-manager-bb756c7c4-wm2cv                  1/1     Running   0
14m
cert-manager-cainjector-c9bb86786-8wrf5       1/1     Running   0
14m
cert-manager-webhook-dd465db99-j2w4x          1/1     Running   0
14m
certificates-68dff9cdd6-kcvml                 1/1     Running   2
(8m43s ago)   9m2s
certificates-68dff9cdd6-rsnsb                 1/1     Running   0
9m2s
cloud-extension-69d48c956c-2s8dt              1/1     Running   3
(8m43s ago)   9m24s
cloud-insights-service-7c4f48b978-7gvlh       1/1     Running   3
(8m50s ago)   9m28s
composite-compute-7d9ff5f68-nxbhl             1/1     Running   0
8m51s
composite-volume-57b4756d64-nl66d             1/1     Running   0
9m13s
credentials-6dbc55f89f-qpzff                  1/1     Running   0
11m
entitlement-67bfb6d7-gl6kp                    1/1     Running   4
(8m33s ago)   9m38s
features-856cc4dccc-mxbdb                     1/1     Running   0
9m20s
fluent-bit-ds-4rtsp                           1/1     Running   0
```

```
6m54s
fluent-bit-ds-9rqll                          1/1     Running   0
6m54s
fluent-bit-ds-w5mp7                          1/1     Running   0
6m54s
graphql-server-7c7cc49776-jz2kn              1/1     Running   0
2m29s
identity-87c59c975-9jpnf                     1/1     Running   0
9m6s
influxdb2-0                                  1/1     Running   0
13m
keycloak-operator-84ff6d59d4-qcnmc           1/1     Running   0
7m1s
krakend-cbf6c7df9-mdtzv                      1/1     Running   0
2m30s
license-5b888b78bf-plj6j                     1/1     Running   0
9m32s
login-ui-846b4664dd-fz8hv                    1/1     Running   0
2m24s
loki-0                                       1/1     Running   0
13m
metrics-facade-779cc9774-n26rw               1/1     Running   0
9m18s
monitoring-operator-974db78f-pkspq           2/2     Running   0
6m58s
nats-0                                       1/1     Running   0
13m
nats-1                                       1/1     Running   0
13m
nats-2                                       1/1     Running   0
13m
nautilus-7bdc7ddc54-49tfn                    1/1     Running   0
7m50s
nautilus-7bdc7ddc54-cwc79                    1/1     Running   0
9m36s
openapi-5584ff9f46-gbrdj                     1/1     Running   0
9m17s
openapi-5584ff9f46-z9mzk                     1/1     Running   0
9m17s
packages-bfc58cc98-lpxq9                     1/1     Running   0
8m58s
polaris-consul-consul-server-0               1/1     Running   0
13m
polaris-consul-consul-server-1               1/1     Running   0
13m
polaris-consul-consul-server-2               1/1     Running   0
```

```
13m
polaris-keycloak-0                          1/1     Running   3
(6m15s ago)   6m56s
polaris-keycloak-1                          1/1     Running   0
4m22s
polaris-keycloak-2                          1/1     Running   0
3m41s
polaris-keycloak-db-0                       1/1     Running   0
6m56s
polaris-keycloak-db-1                       1/1     Running   0
4m23s
polaris-keycloak-db-2                       1/1     Running   0
3m36s
polaris-mongodb-0                           2/2     Running   0
13m
polaris-mongodb-1                           2/2     Running   0
13m
polaris-mongodb-2                           2/2     Running   0
12m
polaris-ui-5ccff47897-8rzgh                 1/1     Running   0
2m33s
polaris-vault-0                             1/1     Running   0
13m
polaris-vault-1                             1/1     Running   0
13m
polaris-vault-2                             1/1     Running   0
13m
public-metrics-6cb7bfc49b-p54xm             1/1     Running   1
(8m29s ago)   9m31s
storage-backend-metrics-5c77994586-kjn48    1/1     Running   0
8m52s
storage-provider-769fdc858c-62w54           1/1     Running   0
8m54s
task-service-9ffc484c5-kx9f4                1/1     Running   3
(8m44s ago)   9m34s
telegraf-ds-bphb9                           1/1     Running   0
6m54s
telegraf-ds-rtsm2                           1/1     Running   0
6m54s
telegraf-ds-s9h5h                           1/1     Running   0
6m54s
telegraf-rs-lbpv7                           1/1     Running   0
6m54s
telemetry-service-57cfb998db-zjx78          1/1     Running   1
(8m40s ago)   9m26s
tenancy-5d5dfbcf9f-vmbxh                    1/1     Running   0
```

```
9m5s
traefik-7b87c4c474-jmgp2                    1/1     Running   0
2m24s
traefik-7b87c4c474-t9k8x                    1/1     Running   0
2m24s
trident-svc-c78f5b6bd-nwdsq                 1/1     Running   0
9m22s
vault-controller-55bbc96668-c6425           1/1     Running   0
11m
vault-controller-55bbc96668-lq9n9           1/1     Running   0
11m
vault-controller-55bbc96668-rfkgg           1/1     Running   0
11m
```

2. (선택 사항) 설치가 완료되었는지 확인하기 위해 을(를) 볼 수 있습니다 `acc-operator` 다음 명령을 사용하여 기록합니다.

```
kubectl logs deploy/acc-operator-controller-manager -n netapp-acc-
operator -c manager -f
```

> `accHost` 클러스터 등록은 마지막 작업 중 하나이며, 클러스터 등록에 실패하면 배포에 실패하지 않습니다. 로그에 클러스터 등록 실패가 표시되는 경우 를 통해 등록을 다시 시도할 수 있습니다 "UI에서 클러스터 워크플로우를 추가합니다" API를 사용합니다.

3. 모든 Pod가 실행되면 설치가 성공적으로 완료되었는지 확인합니다 (`READY` 있습니다 `True`)를 입력하고 Astra Control Center에 로그인할 때 사용할 초기 설치 암호를 받습니다.

```
kubectl get AstraControlCenter -n [netapp-acc or custom namespace]
```

응답:

```
NAME    UUID                                   VERSION     ADDRESS
READY
astra   9aa5fdae-4214-4cb7-9976-5d8b4c0ce27f   22.11.0-82  10.111.111.111
True
```

> UUID 값을 복사합니다. 암호는 입니다 `ACC-` UUID 값 뒤에 옵니다 (`ACC-[UUID]` 또는, 이 예에서는 `ACC-9aa5fdae-4214-4cb7-9976-5d8b4c0ce27f`)를 클릭합니다.

부하 분산을 위한 수신 설정

서비스에 대한 외부 액세스를 관리하는 Kubernetes 수신 컨트롤러를 설정할 수 있습니다. 이 절차에서는 기본값을 사용한 경우 수신 컨트롤러에 대한 설정 예제를 제공합니다 `ingressType: "Generic"` Astra Control Center 사용자 지정 리소스 (`astra_control_center.yaml`)를 클릭합니다. 지정한 경우 이 절차를 사용할 필요가 없습니다 `ingressType: "AccTraefik"` Astra Control Center 사용자 지정 리소스 (`astra_control_center.yaml`)를 클릭합니다.

Astra Control Center를 배포한 후 URL을 사용하여 Astra Control Center를 노출하도록 수신 컨트롤러를 구성해야 합니다.

설치 단계는 사용하는 수신 컨트롤러의 유형에 따라 다릅니다. Astra Control Center는 다양한 수신 컨트롤러 유형을 지원합니다. 이러한 설정 절차에서는 다음과 같은 수신 컨트롤러 유형에 대한 예제 단계를 제공합니다.

- 이스티오 침투
- Nginx 수신 컨트롤러
- OpenShift 수신 컨트롤러

필요한 것

- 필수 요소입니다 "수신 컨트롤러" 이미 배포되어 있어야 합니다.
- 를 클릭합니다 "수신 클래스" 수신 컨트롤러에 해당하는 컨트롤러가 이미 생성되어야 합니다.

Istio 침투에 대한 단계

1. Istio Ingress를 구성합니다.

   이 절차에서는 "기본" 구성 프로파일을 사용하여 Istio를 구축한다고 가정합니다.

2. 수신 게이트웨이에 대해 원하는 인증서 및 개인 키 파일을 수집하거나 생성합니다.

   CA 서명 또는 자체 서명 인증서를 사용할 수 있습니다. 공통 이름은 Astra 주소(FQDN)여야 합니다.

   명령 예:

   ```
   openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout tls.key -out
   tls.crt
   ```

3. 암호를 만듭니다 `tls secret name` 유형 `kubernetes.io/tls` 에서 TLS 개인 키 및 인증서의 경우 `istio-system namespace` TLS 비밀에 설명되어 있습니다.

   명령 예:

   ```
   kubectl create secret tls [tls secret name] --key="tls.key"
   --cert="tls.crt" -n istio-system
   ```

비밀의 이름은 과 일치해야 합니다 `spec.tls.secretName` 에 제공됩니다 `istio-ingress.yaml` 파일.

4. 에 수신 리소스를 배포합니다 `netapp-acc` (또는 사용자 지정 이름) 스키마에 대해 v1 리소스 형식을 사용하는 네임스페이스입니다 (`istio-Ingress.yaml` 이 예에서 사용됨):

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: istio
spec:
  controller: istio.io/ingress-controller
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress
  namespace: [netapp-acc or custom namespace]
spec:
  ingressClassName: istio
  tls:
  - hosts:
    - <ACC address>
    secretName: [tls secret name]
  rules:
  - host: [ACC address]
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: traefik
            port:
              number: 80
```

5. 변경 사항 적용:

```
kubectl apply -f istio-Ingress.yaml
```

6. 수신 상태를 점검하십시오.

```
kubectl get ingress -n [netapp-acc or custom namespace]
```

응답:

```
NAME    CLASS HOSTS              ADDRESS         PORTS   AGE
ingress istio astra.example.com 172.16.103.248  80, 443 1h
```

7. Astra Control Center 설치를 완료합니다.

Nginx 수신 컨트롤러 단계

1. 형식의 암호를 만듭니다 `kubernetes.io/tls` 에서 TLS 개인 키 및 인증서의 경우 `netapp-acc` 에 설명된 대로 (또는 사용자 지정 이름) 네임스페이스를 사용합니다 "TLS 비밀".
2. 수신 리소스를 에 배포합니다 `netapp-acc` (또는 사용자 지정 이름) 스키마에 대해 v1 리소스 형식을 사용하는 네임스페이스입니다 (`nginx-Ingress.yaml` 이 예에서 사용됨):

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: netapp-acc-ingress
  namespace: [netapp-acc or custom namespace]
spec:
  ingressClassName: [class name for nginx controller]
  tls:
  - hosts:
    - <ACC address>
    secretName: [tls secret name]
  rules:
  - host: <ACC address>
    http:
      paths:
        - path:
          backend:
            service:
              name: traefik
              port:
                number: 80
          pathType: ImplementationSpecific
```

3. 변경 사항 적용:

```
kubectl apply -f nginx-Ingress.yaml
```

Nginx 컨트롤러를 이 아닌 배포로 설치하는 것이 좋습니다 `daemonSet`.

OpenShift Ingress 컨트롤러를 위한 단계

1. 인증서를 구입하고 OpenShift 라우트에서 사용할 수 있도록 준비된 키, 인증서 및 CA 파일을 가져옵니다.
2. OpenShift 경로를 생성합니다.

```
oc create route edge --service=traefik --port=web -n [netapp-acc or
custom namespace] --insecure-policy=Redirect --hostname=<ACC address>
--cert=cert.pem --key=key.pem
```

Astra Control Center UI에 로그인합니다

Astra Control Center를 설치한 후 기본 관리자의 암호를 변경하고 Astra Control Center UI 대시보드에 로그인합니다.

단계

1. 브라우저에서 FQDN(을 포함)을 입력합니다 `https://` 접두사)를 입력합니다 `astraAddress` 에 있습니다 `astra_control_center.yaml` CR Astra Control Center를 설치했습니다.
2. 메시지가 표시되면 자체 서명된 인증서를 수락합니다.

   로그인 후 사용자 지정 인증서를 만들 수 있습니다.

3. Astra Control Center 로그인 페이지에서 에 사용한 값을 입력합니다 `email` 인치 `astra_control_center.yaml` CR Astra Control Center를 설치했습니다를 누른 다음 초기 설치 암호를 입력합니다 (`ACC-[UUID]`)를 클릭합니다.

   잘못된 암호를 세 번 입력하면 15분 동안 관리자 계정이 잠깁니다.

4. Login * 을 선택합니다.
5. 메시지가 나타나면 암호를 변경합니다.

   첫 번째 로그인인 경우 암호를 잊어버리고 다른 관리 사용자 계정이 아직 생성되지 않은 경우 에 문의하십시오 "NetApp 지원" 비밀번호 복구 지원을 위해.

6. (선택 사항) 기존의 자체 서명된 TLS 인증서를 제거하고 로 바꿉니다 "인증 기관(CA)에서 서명한 사용자 지정 TLS 인증서".

설치 문제를 해결합니다

에 서비스가 있는 경우 `Error` 상태, 로그를 검사할 수 있습니다. 400 ~ 500 범위의 API 응답 코드를 찾습니다. 이는 고장이 발생한 장소를 나타냅니다.

단계

1. Astra Control Center 운영자 로그를 검사하려면 다음을 입력하십시오.

```
kubectl logs deploy/acc-operator-controller-manager -n netapp-acc-
operator -c manager -f
```

다음 단계

- (선택 사항) 환경에 따라 사후 설치를 완료합니다 "구성 단계".
- 를 수행하여 배포를 완료합니다 "설정 작업".

=
:allow-uri-read:

저작권 정보

상표 정보