Astra Control Center をインストールします

Astra Control Center

NetApp
August 11, 2025

目次

標準の手順で Astra Control Center をインストールします

Astra Control Centerをインストールするには、インストールイメージをダウンロードして次の手順を実行します。この手順を使用して、インターネット接続環境またはエアギャップ環境に Astra コントロールセンターをインストールできます。

Astra Control Centerのインストールプロセスのデモについては、を参照してください "このビデオでは"。

作業を開始する前に

- 環境条件を満たしている： "インストールを開始する前に、 Astra Control Center の導入環境を準備します"。

> 3つ目の障害ドメインまたはセカンダリサイトにAstra Control Centerを導入これは、アプリケーションのレプリケーションとシームレスなディザスタリカバリに推奨されます。

- 正常なサービスを確認：すべてのAPIサービスが正常な状態で使用可能であることを確認します。

```
kubectl get apiservices
```

- *ルーティング可能なFQDN *：使用するAstra FQDNをクラスタにルーティングできることを確認します。つまり、内部 DNS サーバに DNS エントリがあるか、すでに登録されているコア URL ルートを使用しています。
- 証明書マネージャの設定:クラスタに証明書マネージャがすでに存在する場合は、一部の証明書マネージャを実行する必要があります。 "事前に必要な手順" そのため、Astra Control Centerは独自の証明書マネージャのインストールを試みません。デフォルトでは、Astra Control Centerはインストール時に独自の証明書マネージャをインストールします。
- （**ONTAP SAN**ドライバのみ）マルチパスの有効化：ONTAP SANドライバを使用している場合は、すべてのKubernetesクラスタでマルチパスが有効になっていることを確認してください。

また、次の点も考慮する必要があります。

- * NetApp Astra Controlイメージレジストリへのアクセス*：

  Astra Control Provisionerなど、Astra Controlのインストールイメージや機能強化された機能をNetAppイメージレジストリから取得することができます。

  a. レジストリへのログインに必要なAstra ControlアカウントIDを記録します。

     アカウントIDはAstra Control Service Web UIで確認できます。ページ右上の図アイコンを選択し、* APIアクセス*を選択して、アカウントIDを書き留めます。

  b. 同じページから* APIトークンの生成*を選択し、APIトークン文字列をクリップボードにコピーしてエディターに保存します。

  c. Astra Controlレジストリにログインします。

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

- セキュアな通信のためのサービスメッシュをインストール：Astra Controlホストクラスタの通信チャネルは、 "サポートされるサービスメッシュ"。

> Astra Control Centerとサービスメッシュの統合は、Astra Control Centerでのみ実行可能 "インストール" そしてこのプロセスから独立していません。メッシュ環境から非メッシュ環境に戻すことはサポートされていません。

Istioサービスメッシュを使用するには、次の手順を実行する必要があります。

  - を追加します。 `istio-injection:enabled` ラベル にアクセスしてからAstra Control Centerを導入する必要があります。
  - を使用します `Generic` 入力設定 別のイングレスを提供します。 外部ロードバランシング。
  - Red Hat OpenShiftクラスタの場合は、 `NetworkAttachmentDefinition` 関連付けられているすべてのAstra Control Centerネームスペース (`netapp-acc-operator`、 `netapp-acc`、 `netapp-monitoring` アプリケーションクラスタの場合、または置換されたカスタムネームスペースの場合）。

```
cat <<EOF | oc -n netapp-acc-operator create -f -
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: istio-cni
EOF

cat <<EOF | oc -n netapp-acc create -f -
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: istio-cni
EOF

cat <<EOF | oc -n netapp-monitoring create -f -
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: istio-cni
EOF
```

手順

Astra Control Center をインストールするには、次の手順に従います。

- Astra Control Centerをダウンロードして展開します

- [ローカルレジストリを使用する場合は、追加の手順を実行します。]
- [認証要件を持つレジストリのネームスペースとシークレットを設定します]
- Astra Control Center オペレータを設置します
- Astra Control Center を設定します
- Astra Control Center とオペレータのインストールを完了します
- [システムステータスを確認します]
- [ロードバランシング用の入力を設定します]
- Astra Control Center UI にログインします

> Astra Control Centerオペレータ（たとえば、 `kubectl delete -f astra_control_center_operator_deploy.yaml`）Astra Control Centerのインストール中または操作中はいつでも、ポッドを削除しないようにします。

Astra Control Centerをダウンロードして展開します

次のいずれかの場所からAstra Control Centerのイメージをダウンロードします。

- * Astra Controlサービスのイメージレジストリ*：Astra Control Centerのイメージでローカルレジストリを使用しない場合や、NetApp Support Siteからバンドルをダウンロードするよりもこの方法を使用する場合は、このオプションを使用します。
- * NetApp Support Site *：このオプションは、Astra Control Centerのイメージでローカルレジストリを使用する場合に使用します。

Astra Controlイメージレジストリ

1. Astra Control Serviceにログインします。
2. ダッシュボードで、*[Deploy a self-managed instance of Astra Control]*を選択します。
3. 手順に従ってAstra Controlイメージのレジストリにログインし、Astra Control Centerのインストールイメージを取得してイメージを展開します。

NetApp Support Site

1. Astra Control Centerを含むバンドルをダウンロードします (`astra-control-center-[version].tar.gz`）をクリックします "Astra Control Centerのダウンロードページ"。
2. （推奨ですがオプション）Astra Control Centerの証明書と署名のバンドルをダウンロードします (`astra-control-center-certs-[version].tar.gz`）をクリックして、バンドルのシグネチャを確認します。

```
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
```

出力にはと表示されます `Verified OK` 検証が成功したあとに、

3. Astra Control Centerバンドルからイメージを抽出します。

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

ローカルレジストリを使用する場合は、追加の手順を実行します。

Astra Control Centerバンドルをローカルのレジストリにプッシュする場合は、NetApp Astra kubectlコマンドラインプラグインを使用する必要があります。

ネットアップ**Astra kubectl**プラグインをインストール

次の手順を実行して、最新のNetApp Astra kubectlコマンドラインプラグインをインストールします。

作業を開始する前に

ネットアップでは、CPUアーキテクチャやオペレーティングシステム別にプラグインのバイナリを提供しています。このタスクを実行する前に、使用しているCPUとオペレーティングシステムを把握しておく必要があります。

以前のインストールからプラグインがインストールされている場合は、 "最新バージョンがインストールされ

ていることを確認してください" これらの手順を実行する前に。

手順

1. 使用可能なNetApp Astra kubectlプラグインのバイナリを一覧表示します。

   kubectlプラグインライブラリはtarバンドルの一部であり、フォルダに解凍されます `kubectl-astra`。

   ```
   ls kubectl-astra/
   ```

2. オペレーティングシステムとCPUアーキテクチャに必要なファイルを現在のパスに移動し、次の名前に変更します。 `kubectl-astra`：

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

イメージをレジストリに追加する

1. Astra Control Centerバンドルをローカルのレジストリにプッシュする場合は、コンテナエンジンに応じた手順を実行します。

Docker です

a. tarballのルートディレクトリに移動します。次のように表示されます。 `acc.manifest.bundle.yaml` ファイルと次のディレクトリ：

```
acc/
kubectl-astra/
acc.manifest.bundle.yaml
```

b. Astra Control Centerのイメージディレクトリにあるパッケージイメージをローカルレジストリにプッシュします。を実行する前に、次の置換を行ってください `push-images` コマンドを実行します

- <BUNDLE_FILE> をAstra Controlバンドルファイルの名前に置き換えます（`acc.manifest.bundle.yaml`）。
- <MY_FULL_REGISTRY_PATH> をDockerリポジトリのURLに置き換えます。次に例を示します。 "<a href="https://<docker-registry>"" class="bare">https://<docker-registry>"。
- <MY_REGISTRY_USER> をユーザ名に置き換えます。
- <MY_REGISTRY_TOKEN> をレジストリの認証済みトークンに置き換えます。

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

ポドマン

a. tarballのルートディレクトリに移動します。次のファイルとディレクトリが表示されます。

```
acc/
kubectl-astra/
acc.manifest.bundle.yaml
```

b. レジストリにログインします。

```
podman login <YOUR_REGISTRY>
```

c. 使用するPodmanのバージョンに合わせてカスタマイズされた次のいずれかのスクリプトを準備して実行します。<MY_FULL_REGISTRY_PATH> を'サブディレクトリを含むリポジトリのURLに置き換えます

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

```
export REGISTRY=<MY_FULL_REGISTRY_PATH>
export PACKAGENAME=acc
export PACKAGEVERSION=24.02.0-69
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=24.02.0-69
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
```

ⓘ レジストリ設定に応じて、スクリプトが作成するイメージパスは次のようになります。

```
https://downloads.example.io/docker-astra-control-
prod/netapp/astra/acc/24.02.0-69/image:version
```

2. ディレクトリを変更します。

```
cd manifests
```

認証要件を持つレジストリのネームスペースとシークレットを設定します

1. Astra Control Centerホストクラスタのkubeconfigをエクスポートします。

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

> インストールを完了する前に、Astra Control Centerをインストールするクラスタをkubeconfigで指定していることを確認してください。

2. 認証が必要なレジストリを使用する場合は、次の手順を実行する必要があります。

   a. を作成します `netapp-acc-operator` ネームスペース：

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

   b. のシークレットを作成します `netapp-acc-operator` ネームスペース：Docker 情報を追加して次のコマンドを実行します。

> プレースホルダ `your_registry_path` 以前にアップロードした画像の場所と一致する必要があります（例： `[Registry_URL]/netapp/astra/astracc/24.02.0-69` ）。

```
kubectl create secret docker-registry astra-registry-cred -n netapp-acc-operator --docker-server=cr.astra.netapp.io --docker -username=[astra_account_id] --docker-password=[astra_api_token]
```

+

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

+

> シークレットの生成後にネームスペースを削除した場合は、ネームスペースを再作成し、ネームスペースのシークレットを再生成します。

   a. を作成します `netapp-acc` （またはカスタム名）ネームスペース。

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

b. のシークレットを作成します `netapp-acc` （またはカスタム名）ネームスペース。Docker情報を追加し、レジストリの設定に応じて適切なコマンドのいずれかを実行します。

```
kubectl create secret docker-registry astra-registry-cred -n [netapp-acc or custom namespace] --docker-server=cr.astra.netapp.io --docker-username=[astra_account_id] --docker-password=[astra_api_token]
```

```
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]
```

Astra Control Center オペレータを設置します

1. （ローカルレジストリのみ）ローカルレジストリを使用している場合は、次の手順を実行します。

   a. Astra Control Centerオペレータによる導入YAMLを開きます。

   ```
   vim astra_control_center_operator_deploy.yaml
   ```

   注釈付きサンプルYAMLは以下の手順に従います。

   b. 認証が必要なレジストリを使用する場合は、のデフォルト行を置き換えます `imagePullSecrets: []` 次の条件を満たす場合：

   ```
   imagePullSecrets: [{name: astra-registry-cred}]
   ```

   c. 変更 `ASTRA_IMAGE_REGISTRY` をクリックします `kube-rbac-proxy` でイメージをプッシュしたレジストリパスへのイメージ 前の手順。

   d. 変更 `ASTRA_IMAGE_REGISTRY` をクリックします `acc-operator-controller-manager` でイメージをプッシュしたレジストリパスへのイメージ 前の手順。

```
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: ASTRA_IMAGE_REGISTRY/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: ASTRA_IMAGE_REGISTRY/acc-operator:24.02.68
        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
```

2. Astra Control Center オペレータをインストールします。

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

回答例を表示するには展開します。

```
namespace/netapp-acc-operator created
customresourcedefinition.apiextensions.k8s.io/astracontrolcenters.as
tra.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
```

3. ポッドが実行中であることを確認します

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

Astra Control Center を設定します

1. Astra Control Centerカスタムリソース（CR）ファイルを編集します (`astra_control_center.yaml`) アカウント、サポート、レジストリ、およびその他の必要な設定を行うには、次の手順を実行します。

```
vim astra_control_center.yaml
```

注釈付きサンプルYAMLは以下の手順に従います。

2. 次の設定を変更または確認します。

アカウント名

設定	ガイダンス（**Guidance**）	を入力します	例
`accountName`	を変更します `accountName` stringには、Astra Control Centerアカウントに関連付ける名前を指定します。アカウント名は1つだけです。	文字列	`Example`

astraVersion

設定	ガイダンス（**Guidance**）	を入力します	例
`astraVersion`	導入するAstra Control Centerのバージョン。この設定には値があらかじめ入力されているため、対処は不要です。	文字列	`24.02.0-69`

astraitAddress

設定	ガイダンス（**Guidance**）	を入力します	例
`astraAddress`	を変更します `astraAddress` ブラウザで使用するFQDN（推奨）またはIPアドレスを指定して、Astra Control Centerにアクセスします。このアドレスは、データセンターでAstra Control Centerがどのように検出されるかを定義します。このアドレスは、完了時にロードバランサからプロビジョニングしたFQDNまたはIPアドレスと同じです "Astra Control Center の要件"。 注：は使用しないでください `http://` または `https://` をクリックします。この FQDN をコピーしてで使用します 後の手順。	文字列	`astra.example.com`

AutoSupport

このセクションで選択した内容によって、NetAppのプロアクティブサポートアプリケーション、デジタルアドバイザ、およびデータの送信先が決まります。インターネット接続が必要です（ポート442）。サポートデータはすべて匿名化されます。

設定	使用	ガイダンス（**Guidance**）	を入力します	例
`autoSupport.enrolled`	または `enrolled` または `url` フィールドを選択する必要があります	変更 `enrolled` を選択しますAutoSupport `false` インターネットに接続されていないか、または保持されているサイト `true` 接続されているサイト用。の設定 `true` 匿名データをネットアップに送信し、サポートを目的として使用できるようにします。デフォルトの選択はです `false` およびは、サポートデータがネットアップに送信されないことを示します。	ブール値	`false` （デフォルト値）
`autoSupport.url`	または `enrolled` または `url` フィールドを選択する必要があります	このURLは匿名データの送信先を決定します。	文字列	`https://support.netapp.com/asupprod/post/1.0/postAsup`

E メール

設定	ガイダンス（**Guidance**）	を入力します	例
`email`	を変更します `email` デフォルトの初期管理者アドレスを表す文字列。この E メールアドレスをコピーしてで使用します 後の手順。このEメールアドレスは、最初のアカウントがUIにログインする際のユーザ名として使用され、Astra Controlのイベントが通知されます。	文字列	`admin@example.com`

FirstName

設定	ガイダンス（**Guidance**）	を入力します	例
`firstName`	アストラアカウントに関連付けられている初期管理者の名前。ここで使用した名前は、初回ログイン後にUIの見出しに表示されます。	文字列	`SRE`

姓

設定	ガイダンス（**Guidance**）	を入力します	例
`lastName`	アストラアカウントに関連付けられている初期管理者の姓です。ここで使用した名前は、初回ログイン後にUIの見出しに表示されます。	文字列	`Admin`

imageRegistryのこと

このセクションで選択すると、Astraアプリケーションイメージ、Astra Control Center Operator 、Astra Control Center Helmリポジトリをホストするコンテナイメージレジストリが定義されます。

設定	使用	ガイダンス（**Guidance**）	を入力します	例
`imageRegistry.name`	必須	Astra Control Centerの導入に必要なすべてのイメージをホストするAstra Controlイメージレジストリの名前。値は事前に入力されます。ローカルレジストリを設定しないかぎり、アクションは必要ありません。ローカルレジストリの場合は、この既存の値を、前の手順。使用しないでください `http://` または `https://` をレジストリ名に追加します。	文字列	`cr.astra.netapp.io` （デフォルト） `example.registry.com/astra` （ローカルレジストリの例）

設定	使用	ガイダンス（**Guidance**）	を入力します	例
`imageRegistry.secret`	任意。	イメージレジストリでの認証に使用するKubernetesシークレットの名前。値は事前に入力されます。ローカルレジストリを設定し、でそのレジストリに入力した文字列を設定しない限り、アクションは必要ありません。`imageRegistry.name` シークレットが必要です。 重要：認証を必要としないローカルレジストリを使用している場合は、これを削除する必要があります。`secret` ラインの内側 `imageRegistry` または、インストールが失敗します。	文字列	`astra-registry-cred`

ストレージクラス

設定	ガイダンス（**Guidance**）	を入力します	例
`storageClass`	を変更します `storageClass` からの値 `ontap-gold` インストールで必要な別のstorageClassリソースに移動します。コマンドを実行します `kubectl get sc` をクリックして、設定済みの既存のストレージクラスを確認します。Astra Control Provisionerで設定されたストレージクラスのいずれかをマニフェストファイルに入力する必要があります。(`astra-control-center-<version>.manifest`）とをAstra PVSに使用します。設定されていない場合は、デフォルトのストレージクラスが使用されます。 メモ：デフォルトのストレージクラスが設定されている場合は、デフォルトのアノテーションが設定されている唯一のストレージクラスであることを確認してください。	文字列	`ontap-gold`

volumeReclaimPolicyのように指定します

設定	ガイダンス（**Guidance**）	を入力します	オプション（ **Options** ）
`volumeReclaimPolicy`	これにより、AstraのPVSの再利用ポリシーが設定されます。このポリシーをに設定しています `Retain` Astraが削除されたあとに永続的なボリュームを保持このポリシーをに設定しています `Delete` Astraが削除されたあとに永続的ボリュームを削除する。この値が設定されていない場合、PVSは保持されます。	文字列	• `Retain` （デフォルト値） • `Delete`

ingressType

設定	ガイダンス（**Guidance**）	を入力します	オプション（ **Options** ）
`ingressType`	次の入力タイプのいずれかを使用します。 汎用 (`ingressType: "Generic"`）（デフォルト） このオプションは、別の入力コントローラを使用している場合、または独自の入力コントローラを使用する場合に使用します。Astra Control Centerを導入したら、 "入力コントローラ" URLを使用してAstra Control Centerを公開します。 重要：Astra Control Centerでサービスメッシュを使用する場合は、 `Generic` 入力タイプとして入力し、独自の設定を行います。 "入力コントローラ"。 * AccTraefik * (`ingressType: "AccTraefik"`) 入力コントローラを設定しない場合は、このオプションを使用します。これにより、Astra Control Centerが導入されます `traefik` Gateway as a Kubernetes LoadBalancer type serviceの略。 Astra Control Centerは、タイプ「LoadBalancer」のサービスを使用します。 (`svc/traefik` Astra Control Centerの名前空間）で、アクセス可能な外部IPアドレスが割り当てられている必要があります。お使いの環境でロードバランサが許可されていて、設定されていない	文字列	• `Generic` （デフォルト値） • `AccTraefik`

スケールサイズ

設定	ガイダンス（**Guidance**）	を入力します	オプション（ **Options** ）
`scaleSize`	デフォルトでは、Astra で高可用性（HA）が使用されます。 `scaleSize` の `Medium`ほとんどのサービスを`HA`に導入し、冗長性を確保するために複数のレプリカを導入します。を使用 `scaleSize` として `Small`Astraは、消費量を削減するための必須サービスを除き、すべてのサービスのレプリカ数を削減します。ヒント： `Medium` 環境は約100個のポッドで構成されています（一時的なワークロードは含まれません）。100個のポッドは、3つのマスターノードと3つのワーカーノード構成に基づいています）。特にディザスタリカバリのシナリオを検討する場合は、環境で問題 となる可能性があるポッド単位のネットワーク制限に注意してください。	文字列	• `Small` • `Medium` （デフォルト値）

astraitcesScaler

設定	ガイダンス（**Guidance**）	を入力します	オプション（ **Options** ）
`astraResourcesScaler`	AstraeControlCenterリソース制限のスケーリングオプションデフォルトでは、Astra Control CenterはAstra内のほとんどのコンポーネントに対してリソース要求を設定して展開します。この構成により、アプリケーションの負荷と拡張性が高い環境では、Astra Control Centerソフトウェアスタックのパフォーマンスが向上します。ただし、小規模な開発またはテストクラスタを使用するシナリオでは、CRフィールドを使用します `astraResourcesScalar` に設定できます `Off`。これにより、リソース要求が無効になり、小規模なクラスタへの導入が可能になります。	文字列	• `Default` （デフォルト値） • `Off`

追加値

インストール時に既知の問題が表示されないように、Astra Control CenterのCRに次の値を追加します。

```
additionalValues:
    keycloak-operator:
      livenessProbe:
        initialDelaySeconds: 180
      readinessProbe:
        initialDelaySeconds: 180
```

CRD

このセクションで選択した内容によって、Astra Control CenterでのCRDの処理方法が決まります。

設定	ガイダンス（**Guidance**）	を入力します	例
`crds.externalCertManager`	外部証明書マネージャを使用する場合は、変更します `externalCertManager` 終了： `true`。デフォルト `false` Astra Control Centerが、インストール時に独自の証明書マネージャCRDをインストールするようにします。SSDはクラスタ全体のオブジェクトであり、クラスタの他の部分に影響を及ぼす可能性があります。このフラグを使用すると、これらのCRDがAstra Control Centerの外部にあるクラスタ管理者によってインストールおよび管理されることをAstra Control Centerに伝えることができます。	ブール値	`False` （デフォルト値）
`crds.externalTraefik`	デフォルトでは、Astra Control Centerは必要なTraefik CRDをインストールします。SSDはクラスタ全体のオブジェクトであり、クラスタの他の部分に影響を及ぼす可能性があります。このフラグを使用すると、これらのCRDがAstra Control Centerの外部にあるクラスタ管理者によってインストールおよび管理されることをAstra Control Centerに伝えることができます。	ブール値	`False` （デフォルト値）

インストールを完了する前に、構成に適したストレージクラスと入力タイプを選択していることを確認してください。

astra_control_center.yamlの例

```
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: "[cr.astra.netapp.io or your_registry_path]"
    secret: "astra-registry-cred"
  storageClass: "ontap-gold"
  volumeReclaimPolicy: "Retain"
  ingressType: "Generic"
  scaleSize: "Medium"
  astraResourcesScaler: "Default"
  additionalValues:
    keycloak-operator:
      livenessProbe:
        initialDelaySeconds: 180
      readinessProbe:
        initialDelaySeconds: 180
  crds:
    externalTraefik: false
    externalCertManager: false
```

Astra Control Center とオペレータのインストールを完了します

1. 前の手順でまだ行っていない場合は、を作成します `netapp-acc` （またはカスタム）ネームスペース：

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

2. Astra Control Centerでサービスメッシュを使用している場合は、 `netapp-acc` またはカスタムネームスペース：

入力タイプ（`ingressType`）をに設定する必要があります。 `Generic` このコマンドを実行する前に、Astra Control Center CRで確認する必要があります。

```
kubectl label ns [netapp-acc or custom namespace] istio-
injection:enabled
```

3. （推奨） "厳密なMTLを有効にする" Istioサービスメッシュの場合：

```
kubectl apply -n istio-system -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
spec:
  mtls:
    mode: STRICT
EOF
```

4. にAstra Control Centerをインストールします `netapp-acc` （またはカスタムの）ネームスペース：

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

Astra Control Centerのオペレータが環境要件の自動チェックを実行ありません "要件" 原因 でインストールが失敗するか、Astra Control Centerが正常に動作しない可能性があります。を参照してください 次のセクション 自動システムチェックに関連する警告メッセージをチェックします。

システムステータスを確認します

kubectlコマンドを使用すると、システムステータスを確認できます。OpenShift を使用する場合は、同等の OC コマンドを検証手順に使用できます。

手順

1. インストールプロセスで検証チェックに関連する警告メッセージが生成されなかったことを確認します。

```
kubectl get acc [astra or custom Astra Control Center CR name] -n
[netapp-acc or custom namespace] -o yaml
```

その他の警告メッセージは、Astra Control Centerのオペレータログでも報告されます。

2. 自動化された要件チェックによって報告された環境の問題を修正します。

   問題を解決するには、環境がを満たしていることを確認します "要件" （Astra Control Center向け）。

3. すべてのシステムコンポーネントが正常にインストールされたことを確認します。

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

   各ポッドのステータスがになっている必要があります `Running`。システムポッドが展開されるまでに数分かかることがあります。

サンプル応答のために展開

```
acc-helm-repo-5bd77c9ddd-8wxm2              1/1     Running     0
1h
activity-5bb474dc67-8l9ss                   1/1     Running     0
1h
activity-5bb474dc67-qbrtq                   1/1     Running     0
1h
api-token-authentication-6wbj2              1/1     Running     0
1h
api-token-authentication-9pgw6              1/1     Running     0
1h
api-token-authentication-tqf6d              1/1     Running     0
1h
asup-5495f44dbd-z4kft                       1/1     Running     0
1h
authentication-6fdd899858-5x45s             1/1     Running     0
1h
bucketservice-84d47487d-n9xgp               1/1     Running     0
1h
bucketservice-84d47487d-t5jhm               1/1     Running     0
1h
cert-manager-5dcb7648c4-hbldc               1/1     Running     0
1h
cert-manager-5dcb7648c4-nr9qf               1/1     Running     0
1h
cert-manager-cainjector-59b666fb75-bk2tf    1/1     Running     0
1h
cert-manager-cainjector-59b666fb75-pfnck    1/1     Running     0
1h
cert-manager-webhook-c6f9b6796-ngz2x        1/1     Running     0
1h
cert-manager-webhook-c6f9b6796-rwtbn        1/1     Running     0
1h
certificates-5f5b7b4dd-52tnj                1/1     Running     0
1h
certificates-5f5b7b4dd-gtjbx                1/1     Running     0
1h
certificates-expiry-check-28477260-dz5vw    0/1     Completed   0
1h
cloud-extension-6f58cc579c-lzfmv            1/1     Running     0
1h
cloud-extension-6f58cc579c-zw2km            1/1     Running     0
1h
cluster-orchestrator-79dd5c8d95-qjg92       1/1     Running     0
```

```
1h
composite-compute-85dc84579c-nz82f          1/1     Running     0
1h
composite-compute-85dc84579c-wx2z2          1/1     Running     0
1h
composite-volume-bff6f4f76-789nj            1/1     Running     0
1h
composite-volume-bff6f4f76-kwnd4            1/1     Running     0
1h
credentials-79fd64f788-m7m8f                1/1     Running     0
1h
credentials-79fd64f788-qnc6c                1/1     Running     0
1h
entitlement-f69cdbd77-4p2kn                 1/1     Running     0
1h
entitlement-f69cdbd77-hswm6                 1/1     Running     0
1h
features-7b9585444c-7xd7m                   1/1     Running     0
1h
features-7b9585444c-dcqwc                   1/1     Running     0
1h
fluent-bit-ds-crq8m                         1/1     Running     0
1h
fluent-bit-ds-gmgq8                         1/1     Running     0
1h
fluent-bit-ds-gzr4f                         1/1     Running     0
1h
fluent-bit-ds-j6sf6                         1/1     Running     0
1h
fluent-bit-ds-v4t9f                         1/1     Running     0
1h
fluent-bit-ds-x7j59                         1/1     Running     0
1h
graphql-server-6cc684fb46-2x8lr             1/1     Running     0
1h
graphql-server-6cc684fb46-bshbd             1/1     Running     0
1h
hybridauth-84599f79fd-fjc7k                 1/1     Running     0
1h
hybridauth-84599f79fd-s9pmn                 1/1     Running     0
1h
identity-95df98cb5-dvlmz                    1/1     Running     0
1h
identity-95df98cb5-krf59                    1/1     Running     0
1h
influxdb2-0                                 1/1     Running     0
```

```
1h
keycloak-operator-6d4d688697-cfq8b          1/1     Running     0
1h
krakend-5d5c8f4668-7bq8g                    1/1     Running     0
1h
krakend-5d5c8f4668-t8hbn                    1/1     Running     0
1h
license-689cdd4595-2gsc8                    1/1     Running     0
1h
license-689cdd4595-g6vwk                    1/1     Running     0
1h
login-ui-57bb599956-4fwgz                   1/1     Running     0
1h
login-ui-57bb599956-rhztb                   1/1     Running     0
1h
loki-0                                      1/1     Running     0
1h
metrics-facade-846999bdd4-f7jdm             1/1     Running     0
1h
metrics-facade-846999bdd4-lnsxl             1/1     Running     0
1h
monitoring-operator-6c9d6c4b8c-ggkrl        2/2     Running     0
1h
nats-0                                      1/1     Running     0
1h
nats-1                                      1/1     Running     0
1h
nats-2                                      1/1     Running     0
1h
natssync-server-6df7d6cc68-9v2gd            1/1     Running     0
1h
nautilus-64b7fbdd98-bsgwb                   1/1     Running     0
1h
nautilus-64b7fbdd98-djlhw                   1/1     Running     0
1h
openapi-864584bccc-75nlv                    1/1     Running     0
1h
openapi-864584bccc-zh6bx                    1/1     Running     0
1h
polaris-consul-consul-server-0              1/1     Running     0
1h
polaris-consul-consul-server-1              1/1     Running     0
1h
polaris-consul-consul-server-2              1/1     Running     0
1h
polaris-keycloak-0                          1/1     Running     2 (1h
```

```
ago)      1h
polaris-keycloak-1                            1/1     Running     0
1h
polaris-keycloak-db-0                         1/1     Running     0
1h
polaris-keycloak-db-1                         1/1     Running     0
1h
polaris-keycloak-db-2                         1/1     Running     0
1h
polaris-mongodb-0                             1/1     Running     0
1h
polaris-mongodb-1                             1/1     Running     0
1h
polaris-mongodb-2                             1/1     Running     0
1h
polaris-ui-66476dcf87-f6s8j                   1/1     Running     0
1h
polaris-ui-66476dcf87-ztjk7                   1/1     Running     0
1h
polaris-vault-0                               1/1     Running     0
1h
polaris-vault-1                               1/1     Running     0
1h
polaris-vault-2                               1/1     Running     0
1h
public-metrics-bfc4fc964-x4m79                1/1     Running     0
1h
storage-backend-metrics-7dbb88d4bc-g78cj      1/1     Running     0
1h
storage-provider-5969b5df5-hjvcm              1/1     Running     0
1h
storage-provider-5969b5df5-r79ld              1/1     Running     0
1h
task-service-5fc9dc8d99-4q4f4                 1/1     Running     0
1h
task-service-5fc9dc8d99-8l5zl                 1/1     Running     0
1h
task-service-task-purge-28485735-fdzkd        1/1     Running     0
12m
telegraf-ds-2rgm4                             1/1     Running     0
1h
telegraf-ds-4qp6r                             1/1     Running     0
1h
telegraf-ds-77frs                             1/1     Running     0
1h
telegraf-ds-bc725                             1/1     Running     0
```

```
1h
telegraf-ds-cvmxf                              1/1     Running     0
1h
telegraf-ds-tqzgj                              1/1     Running     0
1h
telegraf-rs-5wtd8                              1/1     Running     0
1h
telemetry-service-6747866474-5djnc             1/1     Running     0
1h
telemetry-service-6747866474-thb7r             1/1     Running     1 (1h
ago)      1h
tenancy-5669854fb6-gzdzf                       1/1     Running     0
1h
tenancy-5669854fb6-xvsm2                       1/1     Running     0
1h
traefik-8f55f7d5d-4lgfw                        1/1     Running     0
1h
traefik-8f55f7d5d-j4wt6                        1/1     Running     0
1h
traefik-8f55f7d5d-p6gcq                        1/1     Running     0
1h
trident-svc-7cb5bb4685-54cnq                   1/1     Running     0
1h
trident-svc-7cb5bb4685-b28xh                   1/1     Running     0
1h
vault-controller-777b9bbf88-b5bqt              1/1     Running     0
1h
vault-controller-777b9bbf88-fdfd8              1/1     Running     0
1h
```

4. （オプション） `acc-operator` 進捗状況を監視するログ：

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

> `accHost` クラスタの登録は最後の処理の1つです。登録に失敗しても原因 の導入は失敗しません。ログにクラスタ登録エラーが記録されている場合は、を使用して再度登録を試行できます "UIでクラスタワークフローを追加します" または API 。

5. すべてのポッドが実行中の場合は、インストールが正常に完了したことを確認します (`READY` はです `True`）にアクセスし、Astra Control Centerにログインするときに使用する初期セットアップパスワードを取得します。

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

対応：

```
NAME    UUID                                   VERSION     ADDRESS
READY
astra   9aa5fdae-4214-4cb7-9976-5d8b4c0ce27f  24.02.0-69
10.111.111.111  True
```

UUIDの値をコピーします。パスワードはです `ACC-` 続けてUUIDの値を指定します (`ACC-[UUID]` または、この例では、 `ACC-9aa5fdae-4214-4cb7-9976-5d8b4c0ce27f`）。

ロードバランシング用の入力を設定します

サービスへの外部アクセスを管理するKubernetes入力コントローラを設定できます。これらの手順では、デフォルトのを使用した場合の入力コントローラの設定例を示します `ingressType: "Generic"` Astra Control Centerのカスタムリソース (`astra_control_center.yaml`）。を指定した場合、この手順 を使用する必要はありません `ingressType: "AccTraefik"` Astra Control Centerのカスタムリソース (`astra_control_center.yaml`）。

Astra Control Centerを導入したら、URLを使用してAstra Control Centerを公開するように入力コントローラを設定する必要があります。

セットアップ手順は、使用する入力コントローラのタイプによって異なります。Astra Control Centerは、多くの入力コントローラタイプをサポートしています。ここでは、一部の一般的な入力コントローラタイプの設定手順の例を示します。

作業を開始する前に

- が必要です "入力コントローラ" すでに導入されている必要があります。
- 。 "入力クラス" 入力コントローラに対応するものがすでに作成されている必要があります。

Istio Ingressの手順

1. Istio Ingressを設定します。

   この手順 では、「デフォルト」の構成プロファイルを使用してIstioが導入されていることを前提としています。

2. 入力ゲートウェイに必要な証明書と秘密鍵ファイルを収集または作成します。

   CA署名証明書または自己署名証明書を使用できます。共通名はAstraアドレス（FQDN）である必要があります。

   コマンド例：

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

3. シークレットを作成します `tls secret name` を入力します `kubernetes.io/tls` でTLS秘密鍵と証明書を使用する場合 `istio-system namespace` TLSシークレットで説明されているように、

   コマンド例：

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

   シークレットの名前はと一致する必要があります `spec.tls.secretName` で提供されます `istio-ingress.yaml` ファイル。

4. に入力リソースを配置します `netapp-acc` （またはカスタムネームスペース）。スキーマにはv1リソースタイプを使用します (`istio-Ingress.yaml` は次の例で使用されています）。

```
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. 変更を適用します。

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

6. 入力のステータスを確認します。

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

対応：

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

7. Astra Control Centerのインストールを完了します。

Nginx Ingress Controller の手順

1. タイプのシークレットを作成します `kubernetes.io/tls` でTLSの秘密鍵と証明書を使用する場合 `netapp-acc` （またはカスタム名前付き）ネームスペース。を参照してください "TLS シークレット"。
2. 入力リソースをに配置します `netapp-acc` （またはカスタムネームスペース）。スキーマにはv1リソースタイプを使用します (`nginx-Ingress.yaml` は次の例で使用されています）。

```
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. 変更を適用します。

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

ネットアップでは、nginxコントローラをではなく導入環境としてインストールすることを推奨します `daemonSet`。

OpenShift 入力コントローラの手順

1. 証明書を調達し、 OpenShift ルートで使用できるようにキー、証明書、および CA ファイルを取得します。

2. OpenShift ルートを作成します。

```
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 UI にログインします

Astra Control Centerをインストールしたら、デフォルトの管理者のパスワードを変更し、Astra Control Center UIダッシュボードにログインします。

手順

1. ブラウザで、（を含む）FQDNを入力します `https://` プレフィックス）を使用します `astraAddress` を参照してください `astra_control_center.yaml` CR When（時間） Astra Control Center をインストールした。
2. プロンプトが表示されたら、自己署名証明書を承認します。

   カスタム証明書はログイン後に作成できます。

3. Astra Control Centerのログインページで、に使用した値を入力します `email` インチ `astra_control_center.yaml` CR When（時間） Astra Control Center をインストールしたをクリックし、次に初期セットアップパスワードを入力します (`ACC-[UUID]`）。

   誤ったパスワードを 3 回入力すると、管理者アカウントは 15 分間ロックされます。

4. **[Login]** を選択します。
5. プロンプトが表示されたら、パスワードを変更します。

   初めてログインしたときにパスワードを忘れ、他の管理ユーザアカウントがまだ作成されていない場合は、にお問い合わせください "ネットアップサポート" パスワード回復のサポートを受けるには、

6. （オプション）既存の自己署名 TLS 証明書を削除して、に置き換えます "認証局（ CA ）が署名したカスタム TLS 証明書"。

インストールのトラブルシューティングを行います

いずれかのサービスがにある場合 `Error` ステータスを確認すると、ログを調べることができます。400 ~ 500 の範囲の API 応答コードを検索します。これらは障害が発生した場所を示します。

オプション（ **Options** ）

- Astra Control Center のオペレータログを調べるには、次のように入力します。

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

- Astra Control Center CRの出力を確認するには、次の手順を実行します。

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

別のインストール手順

- * Red Hat OpenShift OperatorHubでインストール*：これを使用 "代替手順" OperatorHubを使用してOpenShiftにAstra Control Centerをインストールするには、次の手順を実行します。
- * Cloud Volumes ONTAP バックエンドを使用してパブリッククラウドにインストール*：ユース "これらの手順に従います" Amazon Web Services（AWS）、Google Cloud Platform（GCP）、またはCloud Volumes ONTAP ストレージバックエンドを使用するMicrosoft AzureにAstra Control Centerをインストールするには、次の手順を実行します。

次のステップ

- （オプション）お使いの環境に応じて、インストール後に実行します "設定手順"。
- "Astra Control Centerをインストールし、UIにログインしてパスワードを変更したら、ライセンスのセットアップ、クラスタの追加、認証の有効化、ストレージの管理、バケットの追加を行うことができます。"。

外部証明書マネージャを設定します

Kubernetesクラスタに証明書マネージャがすでに存在する場合は、Astra Control Centerで独自の証明書マネージャがインストールされないように、いくつかの前提条件となる手順を実行する必要があります。

手順

1. 証明書マネージャがインストールされていることを確認します。

```
kubectl get pods -A | grep 'cert-manager'
```

回答例：

```
cert-manager   essential-cert-manager-84446f49d5-sf2zd              1/1
Running    0      6d5h
cert-manager   essential-cert-manager-cainjector-66dc99cc56-9ldmt   1/1
Running    0      6d5h
cert-manager   essential-cert-manager-webhook-56b76db9cc-fjqrq      1/1
Running    0      6d5h
```

2. の証明書とキーのペアを作成します `astraAddress` FQDN：

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

回答例：

```
Generating a 2048 bit RSA private key
...................+++
.......................................+++
writing new private key to 'tls.key'
```

3. 以前に生成したファイルを使用してシークレットを作成します。

```
kubectl create secret tls selfsigned-tls --key tls.key --cert tls.crt -n
<cert-manager-namespace>
```

回答例：

```
secret/selfsigned-tls created
```

4. を作成します `ClusterIssuer` *とまったく同じ*のファイル。ただし、の名前空間の場所が含まれます `cert-manager` ポッドがインストールされます。

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: astra-ca-clusterissuer
  namespace: <cert-manager-namespace>
spec:
  ca:
    secretName: selfsigned-tls
```

```
kubectl apply -f ClusterIssuer.yaml
```

回答例：

```
clusterissuer.cert-manager.io/astra-ca-clusterissuer created
```

5. を確認します `ClusterIssuer` が正常に起動しました。 `Ready` はである必要があります `True` 次の手順に進む前に、次の手順

```
kubectl get ClusterIssuer
```

回答例：

```
NAME                        READY   AGE
astra-ca-clusterissuer      True    9s
```

6. を実行します "Astra Control Center のインストールプロセス"。があります "Astra Control Centerクラスタ YAMLの必須の設定手順" CRD値を変更して、証明書マネージャが外部にインストールされていることを示します。Astra Control Centerが外部証明書マネージャを認識するように、インストール時にこの手順を完了する必要があります。

著作権に関する情報

商標に関する情報