Astra Control Center をセットアップします

Astra Control Center

NetApp
August 11, 2025

目次

Astra Control Center をセットアップします

Astra Control Center のライセンスを追加します

Astra Control Centerをインストールすると、組み込みの評価用ライセンスがすでにインストールされています。Astra Control Centerを評価する場合は、この手順を省略できます。

新しいライセンスは、Astra Control UIまたはを使用して追加できます "Astra Control API の略"。

Astra Control Centerライセンスは、Kubernetes CPUユニットを使用してCPUリソースを測定し、すべての管理対象Kubernetesクラスタのワーカーノードに割り当てられたCPUリソースを考慮します。ライセンスはvCPUの使用量に基づいています。ライセンスの計算方法の詳細については、を参照してください "ライセンス"。

インストールがライセンス数を超えると、 Astra Control Center は新しいアプリケーションを管理できなくなります。容量を超えるとアラートが表示されます。

既存の評価版またはフルライセンスを更新するには、を参照してください "既存のライセンスを更新する"。

作業を開始する前に

- 新しくインストールしたAstra Control Centerインスタンスへのアクセス。
- 管理者ロールの権限。
- A "ネットアップライセンスファイル" （NLF）。

手順

1. Astra Control Center UI にログインします。
2. 「 * アカウント * > * ライセンス * 」を選択します。
3. 「 * ライセンスの追加 * 」を選択します。
4. ダウンロードしたライセンスファイル（ NLF ）を参照します。
5. 「 * ライセンスの追加 * 」を選択します。

Account>*License* ページには、ライセンス情報、有効期限、ライセンスシリアル番号、アカウント ID 、および使用されている CPU ユニットが表示されます。

評価用ライセンスをお持ちで、AutoSupport にデータを送信していない場合は、Astra Control Centerに障害が発生したときにデータが失われないように、アカウントIDを必ず保存してください。

Astra Control Provisionerを有効にする

Astra Tridentバージョン23.10以降には、Astra Control Provisionerを使用するオプションが用意されています。このオプションを使用すると、ライセンスを取得したAstra

Controlユーザは、高度なストレージプロビジョニング機能にアクセスできます。Astra Control Provisionerは、Astra Trident CSIベースの標準機能に加えて、この拡張機能を提供します。

今後のAstra Controlの更新では、Astra Control ProvisionerがAstra Tridentの代わりにストレージプロビジョニングツールおよびオーケストレータとして使用され、Astra Controlでは必須となります。そのため、Astra ControlのユーザはAstra Control Provisionerを有効にすることを強く推奨します。Astra Tridentは引き続きオープンソースであり、NetAppの新しいCSIやその他の機能でリリース、メンテナンス、サポート、更新されます。

このタスクについて

Astra Control Centerのライセンスを取得していて、Astra Control Provisioner機能を使用する場合は、この手順に従う必要があります。また、Astra Tridentを使用していて、Astra Control Provisionerが提供する追加機能をAstra Controlを使用せずに使用する場合も、この手順に従う必要があります。

どちらの場合も、Astra Trident 24.02ではプロビジョニングツール機能はデフォルトでは有効になっていないため、有効にする必要があります。

作業を開始する前に

Astra Control Provisionerを有効にする場合は、まず次の手順を実行します。

Astra Controlプロビジョニングツールユーザと**Astra Control Center**

- * Astra Control Centerライセンスの取得*： "Astra Control Centerのライセンス" Astra Control Provisionerを有効にし、提供される機能にアクセスするため。
- * Astra Control Center 23.10以降をインストールまたはアップグレード*：Astra Control Provisionerの最新機能（24.02）をAstra Controlで使用する場合は、最新のAstra Control Centerバージョン（24.02）が必要です。
- クラスタに**AMD64**システムアーキテクチャがあることを確認する：Astra Control ProvisionerイメージはAMD64とARM64の両方のCPUアーキテクチャで提供されますが、Astra Control Centerでサポートされるのはamd64のみです。
- レジストリアクセス用の**Astra Control**サービスアカウントを取得：NetApp Support Siteではなく Astra Controlレジストリを使用してAstra Control Provisionerイメージをダウンロードする場合は、"Astra Controlサービスのアカウント"。フォームに必要事項を入力して送信し、BlueXPアカウントを作成すると、Astra Control ServiceのようこそEメールが届きます。
- * Astra Tridentがインストールされている場合は、バージョンが4リリース期間内であることを確認してください*：Astra Tridentがバージョン24.02の4リリース期間内であれば、Astra Control Provisionerを使用してAstra Trident 24.02への直接アップグレードを実行できます。たとえば、Astra Trident 23.04から24.02に直接アップグレードできます。

Astra Control Provisionerのみユーザ

- * Astra Control Centerライセンスの取得*： "Astra Control Centerのライセンス" Astra Control Provisionerを有効にし、提供される機能にアクセスするため。
- * Astra Tridentがインストールされている場合は、バージョンが4リリース期間内であることを確認してください*：Astra Tridentがバージョン24.02の4リリース期間内であれば、Astra Control Provisionerを使用してAstra Trident 24.02への直接アップグレードを実行できます。たとえば、Astra Trident 23.04から24.02に直接アップグレードできます。
- レジストリアクセス用の**Astra Control**サービスアカウントを取得：Astra Control Provisionerイメージをダウンロードするには、レジストリへのアクセスが必要です。使用を開始するには、 "Astra Controlサービスのアカウント"。フォームに必要事項を入力して送信し、BlueXPアカウントを作成すると、Astra Control ServiceのようこそEメールが届きます。

（手順1）Astra Control Provisionerの画像を取得

Astra Control Centerのユーザは、Astra ControlのレジストリまたはNetApp Support Siteメソッドを使用してAstra Control Provisionerイメージを取得できます。Astra Controlを使用せずにAstra Control Provisionerを使用する場合は、レジストリ方式を使用する必要があります。

Astra Controlイメージレジストリ

この手順のコマンドには、Dockerの代わりにPodmanを使用できます。Windows環境を使用している場合は、PowerShellを推奨します。

1. NetApp Astra Controlイメージのレジストリにアクセスします。
   a. Astra Control Service Web UIにログオンし、ページの右上にある図アイコンを選択します。
   b. [API access*]を選択します。
   c. アカウントIDを書き留めます。
   d. 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。
   e. 任意の方法で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>
```

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

2. (カスタムレジストリのみ)イメージをカスタムレジストリに移動するには、次の手順に従います。レジストリを使用していない場合は、 "次のセクション"。
   a. Astra Control Provisionerのイメージをレジストリから取得します。

プルされたイメージは複数のプラットフォームをサポートせず、Linux AMD64など、イメージをプルしたホストと同じプラットフォームのみをサポートします。

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

例

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

   a. 画像にタグを付けます。

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

b. イメージをカスタムレジストリにプッシュします。

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

次のDockerコマンドを実行する代わりに、クレーンコピーを使用できます。
`crane copy cr.astra.netapp.io/astra/trident-acp:24.02.0 <my_custom_registry>/trident-acp:24.02.0`

NetApp Support Site

1. Astra Control Provisionerバンドルをダウンロード (`trident-acp-[version].tar`）をクリックします "Astra Control Centerのダウンロードページ"。
2. （推奨、ただしオプション）Astra Control Centerの証明書とシグネチャバンドル（astra-control-center-certs-[version].tar.gz）をダウンロードして、trident-acp-[version] tar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/AstraControlCenterDockerImages-public.pub -signature certs/trident-acp-[version].tar.sig trident-acp-[version].tar
```

3. Astra Control Provisionerのイメージをロードします。

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

対応：

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

4. 画像にタグを付けます。

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

5. イメージをカスタムレジストリにプッシュします。

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

（ステップ2）Astra TridentでAstra Control Provisionerを有効にする

元のインストール方法で "オペレータ（手動またはHelmを使用）またはtridentctl" そして、元の方法に従って適切な手順を完了します。

Astra Trident運用者

1. "Astra Tridentインストーラをダウンロードして展開"。
2. Astra Tridentをまだインストールしていない場合、または元のAstra Trident環境からオペレータを削除した場合は、次の手順を実行します。
   a. CRDを作成します。

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

   b. Tridentネームスペースを作成（`kubectl create namespace trident`）またはTridentネームスペースがまだ存在することを確認（`kubectl get all -n trident`）。ネームスペースが削除されている場合は、もう一度作成します。
3. Astra Tridentを24.02.0に更新：

> クラスタでKubernetes 1.24以前を実行している場合は、を使用します `bundle_pre_1_25.yaml`。クラスタでKubernetes 1.25以降を実行している場合は、を使用します `bundle_post_1_25.yaml`。

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

4. Astra Tridentが実行されていることを確認します。

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

   対応：

```
NAME      AGE
trident   21m
```

5. [pull-secrets]]シークレットを使用するレジストリがある場合は、Astra Control Provisionerイメージの取得に使用するシークレットを作成します。

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

6. TridentOrchestrator CRを編集し、次の編集を行います。

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

a. Astra Tridentイメージのカスタムレジストリの場所を設定するか、Astra Controlレジストリから取得 (`tridentImage: <my_custom_registry>/trident:24.02.0` または `tridentImage: netapp/trident:24.02.0`）。

b. Astra Control Provisionerを有効にする (`enableACP: true`）。

c. Astra Control Provisionerイメージのカスタムレジストリの場所を設定するか、Astra Controlレジストリから取得 (`acpImage: <my_custom_registry>/trident-acp:24.02.0` または `acpImage: cr.astra.netapp.io/astra/trident-acp:24.02.0`）。

d. もしあなたが 画像プルシークレット この手順では、ここで設定できます。(`imagePullSecrets: - <secret_name>`）。前の手順で設定した名前と同じシークレット名を使用します。

```
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. ファイルを保存して終了します。導入プロセスが自動的に開始されます。

8. operator、deployment、およびReplicaSetsが作成されていることを確認します。

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

Kubernetes クラスタには、オペレータのインスタンスが * 1 つしか存在しないようにしてください。Astra Tridentオペレータを複数の環境に導入することは避けてください。

9. を確認します `trident-acp` コンテナが実行中で、 `acpVersion` はです `24.02.0` ステータス：`Installed`：

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

対応：

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

Tridentctl

1. "Astra Tridentインストーラをダウンロードして展開"。
2. "既存のAstra Tridentがある場合は、そのTridentをホストしているクラスタからアンインストール"。
3. Astra Control Provisionerを有効にしてAstra Tridentをインストール (`--enable-acp=true`）：

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

4. Astra Control Provisionerが有効になっていることを確認します。

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

対応：

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

Helm

1. Astra Trident 23.07.1以前がインストールされている場合は、 "をアンインストールします" オペレータおよびその他のコンポーネント。
2. Kubernetesクラスタが1.24以前を実行している場合は、pspを削除します。

```
kubectl delete psp tridentoperatorpod
```

3. Astra Trident Helmリポジトリを追加します。

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

4. Helmチャートを更新します。

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

対応：

```
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. 画像を一覧表示します。

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

対応：

```
| 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. trident-operator 24.02.0が使用可能であることを確認します。

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

対応：

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

7. 使用 `helm install` これらの設定を含む次のいずれかのオプションを実行します。
   - 導入場所の名前
   - Astra Tridentバージョン
   - Astra Control Provisionerの名前の画像
   - プロビジョニングツールを有効にするフラグ
   - （任意）ローカルレジストリパス。ローカルレジストリを使用している場合は、 "Tridentの画像" 1つのレジストリまたは別のレジストリに配置できますが、すべてのCSIイメージは同じレジストリに配置する必要があります。
   - Tridentネームスペース

オプション（ **Options** ）

- レジストリなしのイメージ

```
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
```

- 1つまたは複数のレジストリ内の画像

```
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
```

を使用できます `helm list` 名前、ネームスペース、グラフ、ステータス、アプリケーションバージョンなどのインストールの詳細を確認するには、次の手順を実行します。とリビジョン番号。

Helmを使用したTridentの導入で問題が発生した場合は、次のコマンドを実行してAstra Tridentを完全にアンインストールします。

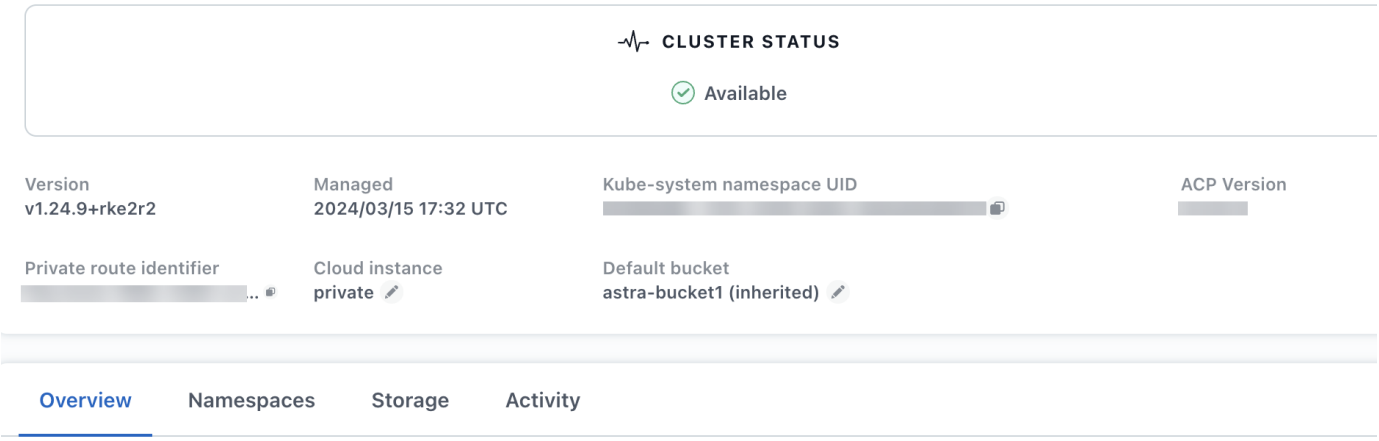
```
./tridentctl uninstall -n trident
```

禁止 "Astra TridentのCRDを完全に削除" Astra Control Provisionerを再度有効にする前に、アンインストールの一環として実行します。

結果

Astra Control Provisionerの機能が有効になっており、実行しているバージョンで使用可能なすべての機能を使用できます。

（Astra Control Centerユーザのみ）Astra Control Provisionerをインストールすると、Astra Control Center UIでプロビジョニングツールをホストしているクラスタに `ACP version` 代わりに `Trident version` フィールドと現在インストールされているバージョン番号。



を参照してください。

- "Astra Tridentのアップグレードに関するドキュメント"

Astra Controlを使用して、クラスタ管理のための環境を準備する

クラスタを追加する前に、次の前提条件を満たしていることを確認する必要があります。また、適性チェックを実行してクラスタをAstra Control Centerに追加する準備ができていることを確認し、必要に応じてkubeconfigクラスタロールを作成する必要があります。

Astra Controlでは、環境や設定に応じて、カスタムリソース（CR）またはkubeconfigで管理されるクラスタを追加できます。

作業を開始する前に

- 環境の前提条件を満たす：環境が次の条件を満たしている "運用環境の要件" （Astra Control Center向け）。
- ワーカーノードの構成: "ワーカーノードの設定" Podがバックエンドストレージと通信できるように、適切なストレージドライバを使用してクラスタを構成します。
- * PSA制限を有効にする*：Kubernetes 1.25以降のクラスタで標準であるポッドセキュリティアドミッション強制が有効になっている場合は、次の名前空間でPSA制限を有効にする必要があります。

- `netapp-acc-operator` ネームスペース：

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

- `netapp monitoring` ネームスペース：

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

- * ONTAP クレデンシャル*：Astra Control Centerを使用してアプリケーションをバックアップおよびリストアするには、バックアップONTAP システムでONTAP クレデンシャルとスーパーユーザーIDを設定する必要があります。

  ONTAP コマンドラインで次のコマンドを実行します。

```
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**が管理するクラスタ要件:これらの要件は、kubeconfigが管理するアプリケーションクラスタに固有のものです。
  - * kubeconfigをアクセス可能にする*: "デフォルトのクラスタkubeconfig" それは "インストール時に設定"。
  - 認証局に関する考慮事項：プライベート認証局（CA）を参照するkubeconfigファイルを使用してクラスタを追加する場合は、 `cluster` kubeconfigファイルのセクションを参照してください。これにより、Astra Controlでクラスタを追加できます。

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

  - **rancher**のみ: Rancher環境でアプリケーションクラスタを管理する場合、rancherから提供されたkubeconfigファイルでアプリケーションクラスタのデフォルトコンテキストを変更して、rancher APIサーバコンテキストではなくコントロールプレーンコンテキストを使用します。これにより、Rancher API サーバの負荷が軽減され、パフォーマンスが向上します。
- * Astra Control Provisionerの要件*：クラスタを管理するには、Astra Tridentコンポーネントを含むAstra Control Provisionerを適切に設定する必要があります。
  - * Astra Trident環境要件の確認*：Astra Control Provisionerをインストールまたはアップグレードする前に、 "サポートされるフロントエンド、バックエンド、およびホスト構成"。
  - * Astra Control Provisioner機能を有効にする*：Astra Trident 23.10以降をインストールして有効にすることを強く推奨します。 "Astra Control Provisionerの高度なストレージ機能"。今後のリリースでは、Astra Control Provisionerが有効になっていない場合、Astra ControlはAstra Tridentをサポートしません。

- ストレージバックエンドの構成:少なくとも1つのストレージバックエンドが "Astra Tridentで設定" クラスタのポリシーを確認してください。
- ストレージクラスの設定：少なくとも1つのストレージクラスが "Astra Tridentで設定" クラスタのポリシーを確認してください。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ます。
- ボリュームスナップショットコントローラを設定し、ボリュームスナップショットクラスをインストールする： "ボリュームSnapshotコントローラのインストール" Astra Controlでスナップショットを作成できるようにします。 "作成" 1つ以上 `VolumeSnapshotClass` Astra Tridentを使用：

資格チェックを実行します

次の資格チェックを実行して、 Astra Control Center にクラスタを追加する準備ができていることを確認します。

手順

1. 実行しているAstra Tridentのバージョンを確認します。

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

Astra Tridentが存在する場合は、次のような出力が表示されます。

```
NAME      VERSION
trident   24.02.0
```

Astra Tridentが存在しない場合は、次のような出力が表示されます。

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

2. 次のいずれかを実行します。
   - Astra Trident 23.01以前を実行している場合は、以下を使用 "手順" Astra Control Provisionerにアップグレードする前に、Astra Tridentの最新バージョンにアップグレードすること。可能です "直接アップグレードを実行する" Astra Tridentがバージョン24.02の4リリース期間内にある場合は、Astra Control Provisioner 24.02をダウンロードします。たとえば、Astra Trident 23.04からAstra Control Provisioner 24.02に直接アップグレードできます。
   - Astra Trident 23.10以降を実行している場合は、Astra Control Provisionerが "有効"。Astra Control Provisionerは、23.10より前のリリースのAstra Control Centerでは機能しません。 "Astra Control Provisionerのアップグレード" 最新の機能にアクセスするには、アップグレードするAstra Control Centerと同じバージョンを使用する必要があります。
3. すべてのポッド（ `trident-acp`）を実行しています：

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

4. サポートされているAstra Tridentドライバをストレージクラスで使用しているかどうかを確認プロビジョ

ニング担当者の名前はとします `csi.trident.netapp.io`。次の例を参照してください。

```
kubectl get sc
```

回答例：

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

クラスタロール**kubeconfig**を作成します。

kubeconfigを使用して管理されるクラスタの場合は、必要に応じて、制限された権限または拡張された権限管理者ロールをAstra Control Centerに対して作成できます。kubeconfigはAstra Control Centerのセットアップですでに設定されているため、これは必須の手順ではありません。 "インストールプロセス"。

この手順を使用すると、次のいずれかのシナリオで環境を環境化する場合に、別のkubeconfigを作成できます。

- 管理対象のクラスタに対するAstra Controlの権限を制限する
- 複数のコンテキストを使用し、インストール時に設定されたデフォルトのAstra Control kubeconfigは使用できません。また、単一のコンテキストを持つ限定されたロールは環境では機能しません。

作業を開始する前に

手順 の手順を実行する前に、管理するクラスタに次の情報があることを確認してください。

- kubectl v1.23以降がインストールされている
- Astra Control Centerを使用して追加および管理するクラスタへのアクセス

> この手順 では、Astra Control Centerを実行しているクラスタにkubectlでアクセスする必要はありません。

- アクティブなコンテキストのクラスタ管理者の権限で管理するクラスタのアクティブなkubeconfigです

手順

1. サービスアカウントを作成します。
   a. という名前のサービスアカウントファイルを作成します `astracontrol-service-account.yaml`。

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

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

b. サービスアカウントを適用します。

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

2. 次のいずれかのクラスタロールを作成し、Astra Controlで管理するクラスタに必要な権限を割り当てます。

クラスタロールの制限

このロールには、Astra Controlでクラスタを管理するために必要な最小限の権限が含まれています。

a. を作成します `ClusterRole` という名前のファイル。例： `astra-admin-account.yaml`。

```
<strong>astra-admin-account.yaml</strong>
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: astra-admin-account
rules:

# Get, List, Create, and Update all resources
# Necessary to backup and restore all resources in an app
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - get
  - list
  - create
  - patch

# Delete Resources
# Necessary for in-place restore and AppMirror failover
- apiGroups:
  - ""
  - apps
  - autoscaling
  - batch
  - crd.projectcalico.org
  - extensions
  - networking.k8s.io
  - policy
  - rbac.authorization.k8s.io
  - snapshot.storage.k8s.io
  - trident.netapp.io
  resources:
  - configmaps
  - cronjobs
  - daemonsets
  - deployments
```

```
    - horizontalpodautoscalers
    - ingresses
    - jobs
    - namespaces
    - networkpolicies
    - persistentvolumeclaims
    - poddisruptionbudgets
    - pods
    - podtemplates
    - 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クラスタの場合のみ） `astra-admin-account.yaml` ファイル：

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

c. クラスタロールを適用します。

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

クラスタロールの拡張

このロールには、Astra Controlで管理するクラスタに対する権限が拡張されています。このロールは、複数のコンテキストを使用し、インストール時に設定されたデフォルトのAstra Control kubeconfigを使用できない場合や、単一のコンテキストを持つ限定されたロールが環境で機能しない場合に使用できます。

次のようになります `ClusterRole` 手順はKubernetesの一般的な例です。ご使用の環境に固有の手順については、ご使用のKubernetesディストリビューションのドキュメントを参照してください。

a. を作成します `ClusterRole` という名前のファイル。例： `astra-admin-account.yaml`。

```
<strong>astra-admin-account.yaml</strong>
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: astra-admin-account
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- nonResourceURLs:
  - '*'
  verbs:
  - '*'
```

b. クラスタロールを適用します。

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

3. サービスアカウントへのクラスタロールバインド用に、クラスタロールを作成します。

a. を作成します `ClusterRoleBinding` という名前のファイルです `astracontrol-clusterrolebinding.yaml`。

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

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: astracontrol-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: astra-admin-account
subjects:
- kind: ServiceAccount
  name: astracontrol-service-account
  namespace: default
```

b. クラスタロールバインドを適用します。

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

4. トークンシークレットを作成して適用します。
   a. という名前のトークンシークレットファイルを作成します。 `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. トークンシークレットを適用します。

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

5. トークンシークレットの名前を `secrets` Array（次の例の最後の行）：

```
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. サービスアカウントのシークレットを一覧表示します（置き換えます） `<context>` インストールに適したコンテキストを使用して、次の操作を行います。

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

出力の末尾は次のようになります。

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

内の各要素のインデックス `secrets` アレイは0から始まります。上記の例では、のインデックスです `astracontrol-service-account-dockercfg-48xhx` は0、のインデックスです `secret-astracontrol-service-account` は1です。出力で、サービスアカウントシークレットのインデックス番号をメモします。このインデックス番号は次の手順で必要になります。

7. 次のように kubeconfig を生成します。
   a. を作成します `create-kubeconfig.sh` ファイル。
   b. 交換してください `TOKEN_INDEX` 次のスクリプトの先頭に正しい値を入力します。

```
<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. コマンドをソースにし、 Kubernetes クラスタに適用します。

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

8. （オプション）クラスタにわかりやすい名前にコバーベキューの名前を変更します。

```
mv kubeconfig-sa YOUR_CLUSTER_NAME_kubeconfig
```

（技術プレビュー）管理対象クラスタ用の**Astra Connector**のインストール

Astra Control Centerで管理されるクラスタは、Astra Connectorを使用して、管理対象クラスタとAstra Control Centerの間の通信を有効にします。管理するすべてのクラスタにAstra Connectorをインストールする必要があります。

Astra Connectorのインストール

KubernetesコマンドとCustom Resource（CR）ファイルを使用してAstra Connectorをインストールします。

このタスクについて

- これらの手順を実行する場合は、Astra Controlで管理するクラスタでこれらのコマンドを実行します。
- 要塞ホストを使用している場合は、要塞ホストのコマンドラインからこれらのコマンドを問題 で実行します。

作業を開始する前に

- Astra Controlで管理するクラスタへのアクセスが必要です。

- クラスタにAstra Connectorオペレータをインストールするには、Kubernetes管理者の権限が必要です。

> Kubernetes 1.25以降のクラスタのデフォルトであるポッドセキュリティアドミッション適用がクラスタに設定されている場合は、適切なネームスペースに対してPSA制限を有効にする必要があります。を参照してください "Astra Controlを使用して、クラスタ管理のための環境を準備する" 手順については、を参照し

手順

1. Astra Controlで管理するクラスタにAstra Connectorオペレータをインストールします。このコマンドを実行すると、ネームスペース `astra-connector-operator` が作成され、設定がネームスペースに適用されます。

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

2. オペレータが設置され、準備ができていることを確認します。

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

3. Astra ControlからAPIトークンを取得を参照してください "Astra Automationのドキュメント" 手順については、を参照し
4. トークンを使用してシークレットを作成します。<API_TOKEN>を、Astra Controlから受け取ったトークンに置き換えます。

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

5. Astra Connectorイメージの取得に使用するDockerシークレットを作成します。括弧<>の値は、環境の情報で置き換えます。

> <ASTRA_CONTROL_ACCOUNT_ID>はAstra Control Web UIで確認できます。Web UIで、ページの右上にある図アイコンを選択し、*[API access]*を選択します。

```
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ファイルを作成してという名前を付ける `astra-connector-cr.yaml`。カッコ内の値を、Astra Controlの環境とクラスタの構成に合わせて更新します。

- <ASTRA_CONTROL_ACCOUNT_ID>：前の手順でAstra Control Web UIから取得。
- <CLUSTER_NAME>：このクラスタをAstra Controlで割り当てる名前。
- <ASTRA_CONTROL_URL>：Astra ControlのWeb UI URL。例：

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

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

7. データを入力した後、 `astra-connector-cr.yaml` 正しい値を持つファイルを作成し、CRを適用します。

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

8. Astra Connectorが完全に導入されたことを確認します。

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

9. クラスタがAstra Controlに登録されたことを確認します。

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

次のような出力が表示されます。

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

10. Astra Control Web UIの*[Clusters]*ページで、管理対象クラスタのリストにクラスタが表示されることを確認します。

クラスタを追加

アプリケーションの管理を開始するには、 Kubernetes クラスタを追加し、コンピューティングリソースとして管理します。Kubernetes アプリケーションを検出するには、 Astra Control Center のクラスタを追加する必要があります。

他のクラスタを Astra Control Center に追加して管理する前に、 Astra Control Center が最初に導入したクラスタを管理することをお勧めします。指標およびトラブルシューティング用の Kubemetrics データとクラスタ関連データを送信するには、最初のクラスタを管理下に配置する必要があります。

作業を開始する前に

- クラスタを追加する前に、必要なを確認し、実行しておきます "前提条件となるタスク"。
- 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ます。

手順

1. ダッシュボードまたはクラスタメニューのいずれかから移動します。
   - リソースサマリの*ダッシュボード*で、クラスタペインから*追加*を選択します。
   - 左側のナビゲーション領域で、*クラスタ*を選択し、クラスタページから*クラスタの追加*を選択します。
2. 表示された*クラスタの追加*ウィンドウで、をアップロードします `kubeconfig.yaml` の内容をファイルまたは貼り付けます `kubeconfig.yaml` ファイル。

。 `kubeconfig.yaml` ファイルには、1つのクラスタのクラスタクレデンシャルのみを含める必要があります*。

自分で作成する場合は `kubeconfig` ファイルには、* 1つの*コンテキストエレメントのみを定義する必要があります。を参照してください "Kubernetes のドキュメント" を参照してください `kubeconfig` ファイル。を使用して、制限されたクラスタロールのkubeconfigを作成した場合 "このプロセスは"この手順では、kubeconfigをアップロードまたは貼り付けてください。

3. クレデンシャル名を指定します。デフォルトでは、クレデンシャル名がクラスタの名前として自動的に入力されます。
4. 「 * 次へ * 」を選択します。

5. このKubernetesクラスタに使用するデフォルトのストレージクラスを選択し、* Next *を選択します。

Astra Control Provisionerで設定され、ONTAPストレージでバックアップされるストレージクラスを選択してください。

6. 情報を確認し、すべてが良好な場合は、「*追加」を選択します。

結果

クラスタが「* discovering *」状態になり、「Healthy *」に変わります。これで、Astra Control Centerを使用してクラスタを管理できるようになりました。

Astra Control Center で管理するクラスタを追加したあと、監視オペレータの配置に数分かかる場合があります。それまでは、通知アイコンが赤に変わり、 * モニタリングエージェントステータスチェック失敗 * イベントが記録されます。この問題は無視してかまいません。問題は、Astra Control Center が正しいステータスを取得したときに解決します。数分経っても問題 が解決しない場合は、クラスタに移動してを実行します `oc get pods -n netapp-monitoring` を開始点として指定します。この問題をデバッグするには、監視オペレータのログを調べる必要があります。

ONTAPストレージバックエンドで認証を有効にする

Astra Control Centerには、ONTAP バックエンドの認証に次の2つのモードがあります。

- クレデンシャルベースの認証：必要な権限を持つONTAP ユーザのユーザ名とパスワード。ONTAP のバージョンとの互換性を最大限に高めるには、adminやvsadminなどの事前定義されたセキュリティログインロールを使用する必要があります。
- 証明書ベースの認証：Astra Control Centerは、バックエンドにインストールされている証明書を使用してONTAP クラスタと通信することもできます。クライアント証明書、キー、および信頼されたCA証明書を使用する（推奨）。

後で既存のバックエンドを更新して、あるタイプの認証から別の方法に移行することができます。一度にサポートされる認証方式は1つだけです。

クレデンシャルベースの認証を有効にします

Astra Control Centerには、クラスタを対象としたクレデンシャルが必要です `admin` ONTAP バックエンドと通信するため。事前定義された標準のロール（など）を使用する必要があります `admin`。これにより、Astra Control Centerの今後のリリースで使用する機能APIが公開される可能性がある、将来のONTAP リリースとの前方互換性が確保されます。

カスタムのセキュリティログインロールはAstra Control Centerで作成して使用できますが、推奨されません。

バックエンド定義の例を次に示します。

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

クレデンシャルがプレーンテキストで保存されるのは、バックエンド定義のみです。クレデンシャルの知識が必要なのは、バックエンドの作成または更新だけです。そのため、Kubernetes管理者またはストレージ管理者が実行するのは管理者専用の操作です。

証明書ベースの認証を有効にします

Astra Control Centerでは、証明書を使用して新規および既存のONTAP バックエンドと通信できます。バックエンド定義には、次の情報を入力する必要があります。

- `clientCertificate`:クライアント証明書。
- `clientPrivateKey`:関連付けられた秘密鍵。
- `trustedCACertificate`:信頼されたCA証明書。信頼された CA を使用する場合は、このパラメータを指定する必要があります。信頼された CA が使用されていない場合は無視してかまいません。

次のいずれかのタイプの証明書を使用できます。

- 自己署名証明書
- サードパーティの証明書

自己署名証明書による認証を有効にします

一般的なワークフローは次の手順で構成されます。

手順

1. クライアント証明書とキーを生成します。生成時に、認証に使用するONTAP ユーザに共通名（CN）を設定します。

```
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. タイプがのクライアント証明書をインストールします `client-ca` とキーをONTAP 入力します。

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

3. ONTAP のセキュリティログインロールが証明書認証方式をサポートしていることを確認します。

```
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. 生成した証明書を使用して認証をテストします。ONTAP 管理LIF>と<vserver name> を管理のIPと名前に置き換えてください。LIFのサービスポリシーがに設定されていることを確認する必要があります `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. 前の手順で得た値を使用して、Astra Control CenterのUIでストレージバックエンドを追加します。

サードパーティの証明書による認証を有効にします

サードパーティの証明書がある場合は、次の手順で証明書ベースの認証を設定できます。

手順

1. 秘密鍵とCSRを生成します。

```
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. CSRをWindows CA（サードパーティCA）に渡し、署名済み証明書を問題 します。
3. 署名済み証明書をダウンロードし、「ontap_signed_cert.crt」という名前を付けます。
4. Windows CA（サードパーティCA）からルート証明書をエクスポートします。
5. このファイルに名前を付けます `ca_root.crt`

   これで、次の3つのファイルが作成されました。

   - 秘密鍵： `ontap_signed_request.key` （これは、ONTAP のサーバ証明書に対応するキーです。サ

ーバ証明書のインストール時に必要です）。

- 署名済み証明書： `ontap_signed_cert.crt` （これは、ONTAP の_server certificate_inとも呼ばれます）。
- ルート**CA**証明書： `ca_root.crt` （これは、ONTAP の_server-ca certificate_inとも呼ばれます）。

6. これらの証明書をONTAP にインストールします。生成してインストールします `server` および `server-ca` ONTAP の証明書。

sample.yamlの展開

```
# 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. パスワードを使用しない通信用に同じホストのクライアント証明書を作成します。Astra Control Center は、このプロセスを使用してONTAP と通信します。
8. クライアント証明書を生成してONTAP にインストールします。

sample.yamlの展開

```
# 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のUIでストレージバックエンドを追加し、次の値を指定します。
   - クライアント証明書：ontap_test_client.pem
   - 秘密鍵：ontap_test_client.key
   - 信頼された**CA**証明書：ontap_signed_cert.crt

ストレージバックエンドを追加します

クレデンシャルまたは証明書認証情報を設定したら、Astra Control Centerに既存のONTAP ストレージバックエンドを追加してリソースを管理できます。

ストレージバックエンドとして Astra Control のストレージクラスタを管理することで、永続ボリューム（ PVS ）とストレージバックエンドの間のリンケージを取得できるだけでなく、追加のストレージ指標も取得できます。

Control Provisionerを有効にしている場合、NetApp SnapMirrorテクノロジを使用する場合、Astra Control

CenterでONTAPストレージバックエンドの追加と管理はオプションです。

手順

1. 左側のナビゲーション領域のダッシュボードで、* Backends *を選択します。
2. 「 * 追加」を選択します。
3. [Add storage backend]ページの[Use existing]セクションで、* ONTAP *を選択します。
4. 次のいずれかを選択します。
   - 管理者のクレデンシャルを使用：ONTAP クラスタ管理IPアドレスと管理者のクレデンシャルを入力します。クレデンシャルはクラスタ全体のクレデンシャルである必要があります。

     ここで入力するクレデンシャルのユーザは、を持っている必要があります `ontapi` ONTAP クラスタのONTAP System Managerで有効になっているユーザログインアクセス方法。SnapMirrorレプリケーションを使用する場合は、アクセス方法が指定された「admin」ロールのユーザクレデンシャルを適用します `ontapi` および `http`、ソースとデスティネーションの両方のONTAP クラスタ。を参照してください "ONTAP ドキュメントの「ユーザーアカウントの管理」を参照してください" を参照してください。

   - 証明書を使用：証明書をアップロードします `.pem` ファイル、証明書キー `.key` ファイルを指定し、必要に応じて認証局ファイルを指定します。
5. 「 * 次へ * 」を選択します。
6. バックエンドの詳細を確認し、 * Manage * を選択します。

結果

バックエンドがに表示されます `online` リストに概要情報を表示します。

バックエンドが表示されるようにページを更新する必要がある場合があります。

バケットを追加します

バケットは、Astra Control UIまたはを使用して追加できます "Astra Control API の略"。アプリケーションと永続的ストレージをバックアップする場合や、クラスタ間でアプリケーションのクローニングを行う場合は、オブジェクトストアバケットプロバイダの追加が不可欠です。Astra Control は、これらのバックアップまたはクローンを、定義したオブジェクトストアバケットに格納します。

アプリケーション構成と永続的ストレージを同じクラスタにクローニングする場合、Astra Controlにバケットを作成する必要はありません。アプリケーションのSnapshot機能にはバケットは必要ありません。

作業を開始する前に

- Astra Control Centerで管理されているクラスタから到達できるバケットを用意します。
- バケットのクレデンシャルがあることを確認します。
- バケットが次のいずれかのタイプであることを確認します。
  - NetApp ONTAP S3

- NetApp StorageGRID S3 の略
- Microsoft Azure
- 汎用 S3

Amazon Web Services（AWS）とGoogle Cloud Platform（GCP）では、汎用のS3バケットタイプを使用します。

Astra Control CenterはAmazon S3を汎用のS3バケットプロバイダとしてサポートしていますが、Astra Control Centerは、AmazonのS3をサポートしていると主張するすべてのオブジェクトストアベンダーをサポートしているわけではありません。

手順

1. 左側のナビゲーション領域で、 * バケット * を選択します。
2. 「 * 追加」を選択します。
3. バケットタイプを選択します。

バケットを追加するときは、正しいバケットプロバイダを選択し、そのプロバイダに適したクレデンシャルを指定します。たとえば、タイプとして NetApp ONTAP S3 が許可され、 StorageGRID クレデンシャルが受け入れられますが、このバケットを使用して原因の以降のアプリケーションのバックアップとリストアはすべて失敗します。

4. 既存のバケット名とオプションの概要 を入力します。

バケット名と概要 はバックアップ先として表示されるため、あとでバックアップを作成する際に選択できます。この名前は、保護ポリシーの設定時にも表示されます。

5. S3 エンドポイントの名前または IP アドレスを入力します。
6. [資格情報の選択*]で、[追加]または[*既存の*を使用]タブのいずれかを選択します。
   - 「*追加」を選択した場合：
     i. Astra Control の他のクレデンシャルと区別するクレデンシャルの名前を入力します。
     ii. クリップボードからコンテンツを貼り付けて、アクセス ID とシークレットキーを入力します。
   - [既存の使用*]を選択した場合：
     i. バケットで使用する既存のクレデンシャルを選択します。
7. 選択するオプション `Add`。

バケットを追加すると、デフォルトのバケットインジケータで1つのバケットがAstra Controlによってマークされます。最初に作成したバケットがデフォルトバケットになります。バケットを追加する際、あとでを選択できます "別のデフォルトバケットを設定する"。

著作権に関する情報

商標に関する情報