Astra Control Automation 22.04ドキュメント

Astra Automation 22.04

NetApp
December 04, 2023

目次

Astra Control Automation 22.04ドキュメント

リリースノート

このリリースについて

このサイトのドキュメントでは、 Astra Control REST API 、および Astra Control の 4 月 2022 （ 22.04 ）リリースで利用可能な関連自動化テクノロジについて説明しています。特に、このリリースの REST API は、対応する 22.04 リリースの Astra Control Center と Astra Control Service に含まれています。

このリリースおよび以前のリリースの詳細については、次のページとサイトを参照してください。

- "Astra Control REST API の新機能"
- "REST のリソースとエンドポイント"
- "Astra Control Center 22.04 のドキュメント"
- "Astra Control Service のマニュアル"
- "旧バージョンの Astra Automation ドキュメント"

Astra Control REST API の新機能

ネットアップでは、 Astra Control REST API を定期的に更新して、新機能、拡張機能、およびバグ修正を提供しています。

2022 年 4 月 26 日（ 2004 年 4 月 22 日）

このリリースでは、 REST API の拡張と更新に加え、セキュリティと管理に関する高度な機能が実装されています。

Astra の新しいリソース

2 つの新しいリソースタイプが追加されました。 * パッケージ * と * アップグレード * です。また、いくつかの既存リソースのバージョンもアップグレードされています。

ネームスペース単位で強化された RBAC

ロールを関連付けられたユーザにバインドする場合は、ユーザがアクセスできるネームスペースを制限できます。詳しくは、 * Role Binding API * のリファレンスおよびを参照してください "RBAC セキュリティ" を参照してください。

バケットの取り外し

不要になったバケットや、正常に機能していないバケットは削除できます。

Cloud Volumes ONTAP のサポート

Cloud Volumes ONTAP がストレージバックエンドとしてサポートされるようになりました。

その他の機能強化

2 つの Astra Control 製品の実装には、次のような機能強化が追加されています。

- Astra Control Center への一般的な入力
- AKS のプライベートクラスタ
- Kubernetes 1.22 のサポート
- VMware Tanzu ポートフォリオのサポート

Astra Control Center および Astra Control Service のドキュメントサイトの「新機能 * 」ページを参照してください。

関連情報

- "Astra Control Center ：新機能"
- "Astra Control Service ：新機能"

2021 年 12 月 14 日（ 21.12 ）

このリリースでは、 REST API の拡張に加え、今後のリリース更新で Astra Control の進化をサポートするためのドキュメント構造の変更が追加されています。

Astra Control の各リリースに対応した、別個の **Astra Automation** のドキュメント

Astra Control の各リリースには、特定のリリースの機能に合わせて拡張およびカスタマイズされた独自の REST API が含まれています。Astra Control REST API の各リリースのドキュメントが、関連する GitHub コンテンツリポジトリに加え、独自の専用 Web サイトで入手できるようになりました。メインのドキュメントサイト "Astra Control Automation の略" 最新リリースのドキュメントは必ず含まれています。を参照してください "旧バージョンの Astra Control Automation のドキュメント" 以前のリリースについては、を参照してください。

REST リソースタイプの拡張

REST リソースタイプの数は、実行フックとストレージバックエンドを重視して拡張が続けられています。新しいリソースには、アカウント、実行フック、フックソース、実行フックオーバーライド、クラスタノード、管理対象のストレージバックエンド、ネームスペース、ストレージデバイス、およびストレージノード。を参照してください "リソース" を参照してください。

NetApp Astra Control Python SDK

NetApp Astra Control Python SDK は、 Astra Control 環境用の自動化コードを簡単に開発できるようにするオープンソースパッケージです。中核となるのは Astra SDK で、 REST API 呼び出しの複雑さを抽象化する一連のクラスが含まれています。また、 Python クラスをラッピングして抽象化することで、特定の管理タスクを実行するツールキットスクリプトもあります。を参照してください "NetApp Astra Control Python SDK" を参照してください。

2021 年 8 月 5 日（ 21.08 ）

このリリースには、新しい Astra 導入モデルの導入と REST API のメジャー拡張が含まれています。

Astra Control Center 導入モデル

このリリースには、パブリッククラウドサービスとして提供される既存の Astra Control Service に加えて、Astra Control Center オンプレミス導入モデルも含まれています。Astra Control Center をサイトにインストールして、ローカルの Kubernetes 環境を管理できます。2 つの Astra Control 導入モデルは同じ REST API を共有しますが、ドキュメントで必要とされるわずかな違いがあります。

REST リソースタイプの拡張

Astra Control REST API からアクセス可能なリソースの数が大幅に増え、多くの新しいリソースがオンプレミスの Astra Control Center の基盤となりました。新しいリソースには、 ASUP 、使用権、機能、ライセンス、設定、 サブスクリプション、バケット、クラウド、クラスタ、管理対象クラスタ、 ストレージバックエンド、およびストレージクラス。を参照してください "リソース" を参照してください。

Astra 環境をサポートする追加のエンドポイント

REST リソースの拡張に加えて、 Astra Control 環境をサポートするための新しい API エンドポイントがいくつか追加されました。

OpenAPI のサポート

OpenAPI エンドポイントは、現在の OpenAPI JSON ドキュメントおよびその他の関連リソースへのアクセスを提供します。

OpenMetrics のサポート

OpenMetrics エンドポイントは、 OpenMetrics リソースを介してアカウントメトリックへのアクセスを提供します。

2021 年 4 月 15 日（ 21.04 ）

このリリースには、次の新機能と機能拡張が含まれています。

REST API の導入

Astra Control REST API は、 Astra Control Service と組み合わせて使用できます。REST テクノロジと現在のベストプラクティスに基づいて作成されています。この API は、 Astra 環境を自動化するための基盤となり、次の機能とメリットが含まれています。

リソース

REST リソースには 14 種類あります。

API トークンアクセス

REST API には、 Astra Web ユーザインターフェイスで生成できる API アクセストークンを使用してアクセスできます。API トークンを使用して、 API に安全にアクセスできます。

収集のサポート

リソースコレクションへのアクセスに使用できる豊富なクエリパラメータセットがあります。フィルタ、ソート、ページ付けなどの処理がサポートされます。

既知の問題

Astra Control REST API に関連する現在のリリースの既知の問題をすべて確認しておく必要があります。ここでは、この製品の正常な使用を妨げる可能性のある既知の問題について記載します。

> Astra Control REST API の 22.04 リリースでは、新しい既知の問題はありません。以下に記載する問題は以前のリリースで見つかったもので、現在のリリースにも該当します。

バックエンドストレージノード内の一部のストレージデバイスが検出されていません

ストレージノードに定義されているストレージデバイスを取得するREST API呼び出しを発行したときに、すべてのデバイスが返されるわけではありません。

Astra Control REST API の概要

Astra Control Center と Astra Control Service は、 Curl などのプログラミング言語やユーティリティを使用して直接アクセスできる共通の REST API を提供します。API の主な特長とメリットを以下に示します。

> REST API にアクセスするには、最初に Astra Web ユーザインターフェイスにサインインして、 API トークンを生成する必要があります。トークンは各 API 要求に含める必要があります。

REST テクノロジを基盤として構築

REST テクノロジを使用して Astra Control API が作成され、最新のベストプラクティスが適用されている。中核となるテクノロジは、 HTTP 、 JSON 、 RBAC です。

2 つの **Astra Control** 導入モデルをサポート

Astra Control Service はパブリッククラウド環境で使用され、 Astra Control Center はオンプレミス環境に使用されます。これらの両方の導入モデルをサポートする REST API が 1 つあります。

REST エンドポイントリソースとオブジェクトモデルの間のマッピングをクリアします

リソースへのアクセスに使用される外部の REST エンドポイントは、 Astra サービスによって内部的に管理されている整合性のあるオブジェクトモデルにマッピングされます。オブジェクトモデルは、エンティティと関係（ ER ）のモデリングを使用して設計され、 API のアクションと応答を明確に定義できます。

豊富なクエリパラメータ

REST API には、リソースコレクションへのアクセスに使用できる豊富なクエリパラメータが用意されています。フィルタ、ソート、ページ付けなどの処理がサポートされます。

Astra Control Web UI とのアライメント

設計された Astra Web ユーザインターフェイスは REST API に対応しているため、 2 つのアクセスパスとユーザエクスペリエンスの間で一貫性があります。

堅牢なデバッグおよび問題の特定データ

Astra Control REST API は、システムイベントやユーザ通知など、堅牢なデバッグおよび問題の特定機能を提供します。

ワークフロープロセス

自動化コードの開発を支援する一連のワークフローが用意されています。ワークフローは、インフラと管理の 2 つの主要なカテゴリに分類されています。

高度なオートメーション技術の基盤

REST API に直接アクセスするだけでなく、 REST API をベースとするその他の自動化テクノロジも使用できます。

Astra ファミリーのドキュメントの一部

Astra Control Automation のドキュメントは、 Astra ファミリーの大きなドキュメントの一部です。を参照してください "Astra のドキュメント" を参照してください。

はじめに

作業を開始する前に

次の手順を確認して、 Astra Control REST API の使用を開始するための準備を簡単に行うことができます。

Astra アカウントのクレデンシャルが必要

Astra のクレデンシャルが必要になるのは、 Astra Web ユーザインターフェイスにサインインして、 API トークンを生成する場合です。Astra Control Center を使用すると、これらの資格情報をローカルで管理できます。Astra Control Service を使用すると、アカウントの資格情報に * Auth0 * サービスからアクセスできます。

Kubernetes の基本概念を理解する

Kubernetes のいくつかの基本概念を理解しておく必要があります。を参照してください "Kubernetes の基本概念" を参照してください。

REST の概念と実装を確認

必ず確認してください "コア REST の実装" REST の概念と、 Astra Control REST API の設計に関する詳細については、を参照してください。

詳細はこちらをご覧ください

に示す追加情報のリソースを確認しておく必要があります "その他のリソース"。

API トークンを取得します

Astra Control REST API を使用するには、 Astra API トークンを取得する必要があります。

はじめに

API トークンは、 Astra の呼び出し元を識別し、すべての REST API 呼び出しに含める必要があります。

- Astra Web ユーザインターフェイスを使用して、 API トークンを生成できます。
- トークンを使用して伝送されるユーザ ID は、トークンを作成するユーザによって決まります。
- トークンは 'Authorization'HTTP 要求ヘッダーに含める必要があります
- トークンは作成後に期限切れになることはありません。
- トークンは、 Astra の Web ユーザインターフェイスから取り消すことができます。

関連情報

- "API トークンを取り消します"

Astra API トークンを作成

Astra API トークンを作成する手順を次に示します。

作業を開始する前に

Astra アカウントのクレデンシャルが必要です。

このタスクについて

このタスクは、 Astra Web インターフェイスで API トークンを生成します。また、 API 呼び出しの際にも必要なアカウント ID を取得する必要があります。

手順

1. アカウントのクレデンシャルを使用して Astra にサインインします。

   Astra Control Service の次のサイトにアクセスします。 "https://astra.netapp.io"

2. ページの右上にある図のアイコンをクリックし、 * API access * を選択します。
3. ページで [API トークンの生成] をクリックし、ポップアップウィンドウで [API トークンの生成] をクリックします。
4. アイコンをクリックしてトークン文字列をクリップボードにコピーし、エディタに保存します。
5. 同じページにあるアカウント ID をコピーして保存します。

完了後

Curl またはプログラミング言語を使用して Astra Control REST API にアクセスする場合は、 API ベアラトークンを HTTP'Authorization' 要求ヘッダーに含める必要があります。

Hello world

ワークステーションの CLI でシンプルな Curl コマンドを問題して、 Astra Control REST API の使用を開始し、利用可能かどうかを確認できます。

作業を開始する前に

Curl ユーティリティがローカルワークステーションで使用可能である必要があります。また、 API トークンと関連付けられているアカウント識別子も必要です。を参照してください "API トークンを取得します" を参照してください。

カールの例

次の Curl コマンドは、 Astra ユーザのリストを取得します。図に示すように、適切な <account_ID> と <api_ctoken> を指定します。

```
curl --location --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/core/v1/users' --header
'Content-Type: application/json' --header 'Authorization: Bearer
<API_TOKEN>'
```

JSON 出力例

```
{
  "items": [
    [
    "David",
    "Peterson",
    "844ec6234-11e0-49ea-8434-a992a6270ec1"
    ],
    [
      "Scott",
      "Morris",
      "2a3e227c-fda7-4145-a86c-ed9aa0183a6c"
    ]
  ],
  "metadata": {}
}
```

ワークフローを使用する準備をします

実際の環境で使用する前に、 Astra ワークフローの構成と形式を理解しておく必要があります。

はじめに

a_workflow_ は、特定の管理タスクまたは目標を達成するために必要な 1 つ以上のステップのシーケンスです。Astra Control ワークフローの各手順は、次のいずれかです。

- REST API 呼び出し（ cURL や JSON の例などの詳細を含む）
- 別の Astra ワークフローの呼び出し
- その他の関連タスク（必要な設計決定の実行など）

ワークフローには、各タスクを実行するために必要な主要な手順とパラメータが含まれています。自動化環境をカスタマイズするための出発点となります。

共通の入力パラメータ

以下に記載する入力パラメータは、 REST API 呼び出しを示すために使用するすべての cURL サンプルに共通しています。

> これらの入力パラメータは汎用的に必要なため、個々のワークフローでは詳しく説明していません。特定のカールの例に追加の入力パラメータが使用される場合は、「 * その他の入力パラメータ * 」セクションで説明します。

パスパラメータ

すべての REST API 呼び出しで使用されるエンドポイントパスには、次のパラメータが含まれています。も参照してください "URL 形式" を参照してください。

アカウント **ID**

これは、 API 処理を実行する Astra アカウントを識別する UUIDv4 値です。を参照してください "API トークンを取得します" アカウント ID の検索の詳細については、を参照してください。

要求ヘッダー

REST API 呼び出しに応じて、いくつかの要求ヘッダーを含める必要があります。

承認

ワークフロー内のすべての API 呼び出しで、ユーザを識別するための API トークンが必要です。トークンは 'Authorization' 要求ヘッダーに含める必要がありますを参照してください "API トークンを取得します" API トークンの生成の詳細については、を参照してください。

コンテンツタイプ

要求の本文に JSON が含まれている HTTP POST 要求と PUT 要求では、 Astra リソースに基づいてメディアタイプを宣言する必要があります。たとえば ' 管理対象アプリケーションのスナップショットを作成するときに 'Content-Type:application/stra-pappSnap+JSOb' というヘッダーを含めることができます

同意します

アストラリソースに基づいて、応答で想定されるコンテンツの特定のメディアタイプを宣言できます。たとえば ' 管理対象アプリケーションのバックアップを一覧表示するときに 'Accept:application/Astra-pappBackup+JSOb' というヘッダーを含めることができますただし、簡単にするために、ワークフロー内のカールサンプルはすべてのメディアタイプに対応しています。

トークンと識別子の表示

cURL の例で使用される API トークンおよびその他の ID 値は不透明で、認識不能な意味はありません。サンプルの読みやすさを向上させるために、実際のトークンと ID 値は使用されません。代わりに、小さい予約済みキーワードが使用されます。これには次のような利点があります。

- cURL と JSON のサンプルは、より明確でわかりやすくなっています。
- すべてのキーワードは角かっこと大文字で同じ形式を使用するため、挿入または抽出する場所とコンテンツをすばやく識別できます。
- 元のパラメータをコピーして実際の配置で使用することはできないため、値は失われません。

次に、 curl の例で使用される一般的な予約済みキーワードの一部を示します。このリストはすべてを網羅しているわけではなく、必要に応じてその他のキーワードが使用されていその意味はコンテキストに基づいて明確になる必要があります。

キーワード	を入力します	説明
<account_ID>	パス	API 処理を実行するアカウントを識別する UUIDv4 値。
<api_ctoken> のように指定します	ヘッダー	発信者を識別および認可するベアラトークン。

キーワード	を入力します	説明
<managed_app_ID>	パス	API 呼び出しの管理対象アプリケーションを識別する UUIDv4 値。

ワークフローのカテゴリ

導入モデルに応じて、幅広い種類の Astra ワークフローが用意されています。Astra Control Center を使用している場合は、インフラワークフローから始めて、管理ワークフローに進みます。Astra Control Service を使用すると、通常は管理ワークフローに直接移動できます。

> ワークフロー内の cURL のサンプルでは、 Astra Control Service の URL を使用します。オンプレミスの Astra Control Center を使用している場合は、環境に応じて URL を変更する必要があります。

インフラワークフロー

これらのワークフローは、クレデンシャル、バケット、ストレージバックエンドなどの Astra インフラを処理します。Astra Control Center で必要ですが、ほとんどの場合は Astra Control Service でも使用できます。このワークフローでは、マネージドクラスタの構築と保守に必要なタスクを中心に説明します。

管理ワークフロー

これらのワークフローは、管理対象クラスタを作成したあとに使用できます。このワークフローでは、バックアップ、リストア、管理対象アプリケーションのクローニングなど、アプリケーションの保護とサポートの処理に重点を置いています。

Kubernetes の基本概念

Kubernetes の概念については、 Astra REST API の使用時にいくつか関連するものがあります。

オブジェクト

Kubernetes 環境で管理されるオブジェクトは、クラスタの構成を表す永続的なエンティティです。これらのオブジェクトのことから、クラスタワークロードを含むシステムの状態がわかります。

ネームスペース

ネームスペースは、単一のクラスタ内でリソースを分離する手法を提供します。この組織構造は、作業、ユーザ、およびリソースのタイプを分割する場合に便利です。namespace scope_を 持つオブジェクトはネームスペース内で一意である必要があり、 _cluster scope_を 持つオブジェクトはクラスタ全体で一意である必要があります。

ラベル

ラベルは Kubernetes オブジェクトに関連付けることができます。キーと値のペアを使用して属性を記述します。また、 Kubernetes の中核的な処理には含まれない、組織には役立つ任意の組織をクラスタに適用できます。

コア REST の実装

REST Web サービス

Representational State Transfer （ REST ）は、分散 Web アプリケーションの作成に使用される形式です。Web サービス API の設計に適用されることで、サーバベースのリソースを公開し、その状態を管理するための一連の主流テクノロジとベストプラクティスが確立されます。REST はアプリケーション開発のための一貫した基盤を提供しますが、各 API の詳細は設計内容に応じて異なる場合があります。ライブ環境で使用する前に、 Astra Control REST API の特性を理解しておく必要があります。

リソースと状態の表示

リソースは、 Web ベースシステムの基本コンポーネントです。REST Web サービスアプリケーションを作成する場合、設計の早い段階で次の作業を行います。

- システムまたはサーバベースのリソースの識別

  すべてのシステムは、リソースを使用および管理します。リソースには、ファイル、ビジネストランザクション、プロセス、管理エンティティなどがあります。REST Web サービスに基づいてアプリケーションを設計する際に行う最初の作業の 1 つは、リソースを識別することです。

- リソースの状態および関連する状態操作の定義

  リソースの状態の数は有限で、リソースは必ずそのいずれかの状態にあります。状態、および状態の変化に影響する関連操作を明確に定義する必要があります。

URI エンドポイント

すべての REST リソースは、明確に定義されたアドレス指定方式を使用して定義および使用可能にする必要があります。リソースが置かれているエンドポイントは、 Uniform Resource Identifier （ URI ）で識別されます。URI は、ネットワーク内の各リソースに一意の名前を作成するための一般的なフレームワークです。Uniform Resource Locator （ URL ）は、リソースを識別してアクセスするために Web サービスで使用される URI の一種です。リソースは通常、ファイルディレクトリに似た階層構造で公開されます。

HTTP メッセージ

Hypertext Transfer Protocol （ HTTP ）は、 Web サービスのクライアントとサーバがリソースに関する要求と応答のメッセージを交換する際に使用するプロトコルです。Web サービスアプリケーションの設計の一環として、 HTTP メソッドはリソースおよび対応する状態管理アクションにマッピングされます。HTTP はステートレスです。したがって、関連する一連の要求と応答を 1 つのトランザクションの一部として関連付けるには、要求と応答のデータフローで伝送される HTTP ヘッダーに追加情報を含める必要があります。

JSON 形式

Web サービスのクライアントとサーバの間で情報を構造化して転送する方法は複数ありますが、最も広く使用されているのは JavaScript Object Notation （ JSON ）です。JSON は、単純なデータ構造をプレーンテキストで表すための業界標準であり、リソースについての状態情報の転送に使用されます。Astra Control REST

API では、 JSON を使用して、各 HTTP 要求と応答の本文で伝送されるデータをフォーマットします。

リソースとコレクション

Astra Control REST API を使用すると、リソースインスタンスとリソースインスタンスのコレクションにアクセスできます。

> 概念的には REST * リソース * は、オブジェクト指向プログラミング（ OOP ）言語およびシステムで定義される * オブジェクト * に似ています。これらの用語が同じ意味で使用されることもあります。ただし一般には、外部 REST API のコンテキストで使用する場合は「リソース」が推奨され、サーバに格納される対応するステートフルインスタンスデータには「オブジェクト」が使用されます。

Astra リソースの属性

Astra Control REST API は、 RESTful 設計の原則に準拠しています。各アストラリソースインスタンスは、明確に定義されたリソースタイプに基づいて作成されます。同じタイプのリソースインスタンスのセットを * 集合 * と呼びます。API 呼び出しは、個々のリソースまたはリソースの集合に対して機能します。

リソースタイプ

Astra Control REST API に含まれるリソースタイプには、次の特徴があります。

- すべてのリソースタイプはスキーマを使用して定義されます（通常は JSON 形式）。
- すべてのリソーススキーマには、リソースタイプとバージョンが含まれています
- リソースタイプはグローバルに一意です

リソースインスタンス

Astra Control REST API から使用できるリソースインスタンスには、次のような特徴があります。

- リソースインスタンスは、単一のリソースタイプに基づいて作成されます
- リソースタイプは、 ［ メディアタイプ ］ の値を使用して示されます
- インスタンスは、アストラサービスによって管理されるステートフルデータで構成されます
- 各インスタンスには、一意で長く存続する URL を使用してアクセスできます
- リソースインスタンスが複数のリプレゼンテーションを持つことができる場合は、異なるメディアタイプを使用して必要なリプレゼンテーションを要求できます

リソースコレクション

Astra Control REST API で使用できるリソース収集には、次のような特徴があります。

- 単一のリソースタイプの一連のリソースインスタンスをコレクションと呼びます
- 一連のリソースには、一意で一時的な URL があります

インスタンス **ID**

すべてのリソースインスタンスには、作成時に識別子が割り当てられます。この識別子は 128 ビットの UUIDv4 値です。割り当てられた UUIDv4 の値は、グローバルに一意で変更できません。新しいインスタンスを作成する API 呼び出しを発行すると、関連付けられた ID を持つ URL が HTTP 応答の「 Location 」ヘッダ

ーで呼び出し元に返されます。リソースインスタンスを以降の呼び出しで参照する際には、この識別子を抽出して使用できます。

> リソース識別子は、コレクションで使用される主キーです。

Astra のリソースに共通の構造

すべてのアストラ制御リソースは、共通の構造を使用して定義されます。

共通のデータ

すべてのアストラリソースには、次の表に示すキーと値が含まれています。

キーを押します	説明
を入力します	グローバルに一意なリソースタイプ。 * リソースタイプ * と呼ばれます。
バージョン	* リソースバージョン * と呼ばれるバージョン識別子。
ID	グローバル一意識別子。 * リソース識別子 * と呼ばれます。
メタデータ	ユーザラベルやシステムラベルなどのさまざまな情報を含む JSON オブジェクト。

メタデータオブジェクト

各アストラリソースに含まれるメタデータ JSON オブジェクトには、次の表に示すキーと値が含まれています。

キーを押します	説明
ラベル	リソースに関連付けられているクライアント指定のラベルの JSON 配列。
作成タイムスタンプ	リソースが作成されたときのタイムスタンプを含む JSON 文字列。
modificationTimestamp	リソースの最終変更日時を示す ISO-8601 形式のタイムスタンプを含む JSON 文字列。
作成者	リソースを作成したユーザ ID の UUIDv4 識別子を含む JSON 文字列。リソースが内部システムコンポーネントによって作成され、作成するエンティティに関連付けられている UUID がない場合は、「 * null * UUID 」が使用されます。

リソースの状態

選択されたリソースは ' ライフサイクルの移行とアクセス制御のオーケストレーションに使用される価値を示します

HTTP の詳細

Astra Control REST API は、 HTTP および関連パラメータを使用して、リソースとコレクションに対して操作を行います。HTTP 実装の詳細を以下に示します。

API トランザクションと CRUD モデル

Astra Control REST API は、明確に定義された操作と状態の移行を伴うトランザクションモデルを実装します。

要求と応答の API トランザクション

すべての REST API 呼び出しは、 Astra サービスへの HTTP 要求として実行されます。要求ごとに、関連付けられた応答がクライアントに生成されます。この要求と応答のペアで API トランザクションを使用できます。

CRUD 操作モデルのサポート

Astra Control REST API で使用できる各リソースインスタンスとコレクションには、 *CRUD * モデルに基づいてアクセスします。4 つの操作があり、それぞれが単一の HTTP メソッドにマッピングされます。処理は次のとおりです。

- 作成
- 読み取り
- 更新
- 削除

一部の Astra リソースでは、これらの操作の一部のみがサポートされています。を確認しておきます "API リファレンス" 特定の API 呼び出しに関する詳細については、を参照してください。

HTTP メソッド

次の表に、 API でサポートされる HTTP メソッドまたは動詞を示します。

メソッド	CRUD	説明
取得	読み取り	リソースインスタンスまたはコレクションのオブジェクトプロパティを取得します。これは、コレクションとともに使用される場合、 * list * 演算と見なされます。
投稿（ Post ）	作成	入力パラメータに基づいて新しいリソースインスタンスを作成します。長期 URL は 'Location' 応答ヘッダーで返されます
PUT	更新	指定した JSON 要求の本文でリソースインスタンス全体を更新します。ユーザが変更できないキー値は保持されます。
削除	削除	既存のリソースインスタンスを削除します。

要求と応答のヘッダー

次の表に、 Astra Control REST API で使用される HTTP ヘッダーを示します。

を参照してください "RFC 7232" および "RFC 7233" を参照してください。

ヘッダー	を入力します	使用上の注意
同意します	リクエスト	値が「 * / * 」または指定されていない場合、「 application/json 」は Content-Type 応答ヘッダーで返されます。値が Astra リソースのメディアタイプに設定されている場合、同じメディアタイプが Content-Type ヘッダーに返されます。
承認	リクエスト	ユーザーの API キーを使用した Bearer トークン。
コンテンツタイプ	応答	「 Accept 」要求ヘッダーに基づいて返されます。
ETag	応答	RFC 7232 で定義されたとおりに、成功したに含まれます。値は、 JSON リソース全体の MD5 値を 16 進数で表したものです。
if-match	リクエスト	セクション 3.1 「 RFC 7232 」で説明されているように、 * PUT * 要求をサポートする前提条件の要求ヘッダーが実装されています。
If-Modified-Since の略	リクエスト	セクション 3.4 「 RFC 7232 」に記載されている前提条件の要求ヘッダーと、 *PUT * 要求のサポート。
If - unmodified- since	リクエスト	セクション 3.4 「 RFC 7232 」に記載されている前提条件の要求ヘッダーと、 *PUT * 要求のサポート。
場所	応答	新しく作成されたリソースの完全な URL が含まれます。

クエリパラメータ

リソースコレクションでは、次のクエリパラメータを使用できます。を参照してください "コレクションの操作" を参照してください。

クエリパラメータ	説明
含める	コレクションを読み取るときに返すフィールドが含まれています。
フィルタ	コレクションの読み取り時にリソースが返されるために一致しなければならないフィールドを示します。
OrderBy	コレクションの読み取り時に返されるリソースのソート順序を指定します。
制限（ Limit ）	コレクションの読み取り時に返されるリソースの最大数を制限します。
スキップします	コレクションの読み取り時に、パスオーバーおよびスキップするリソースの数を設定します。
カウント	メタデータオブジェクトで返されるリソースの総数。

HTTP ステータスコード

Astra Control REST API で使用される HTTP ステータスコードを次に示します。

> Astra Control REST API は、 HTTP API * 標準の * 問題の詳細も使用します。を参照してください "診断とサポート" を参照してください。

コード	意味	説明
200	わかりました	新しいリソースインスタンスを作成しない呼び出しが成功したことを示します。
201	作成済み	オブジェクトが作成され、場所の応答ヘッダーにオブジェクトの一意の識別子が含まれている。
204	コンテンツがありません	要求は成功しましたが、コンテンツが返されませんでした。
400	無効な要求です	要求の入力が認識されないか不適切です。
401	権限がありません	ユーザは認証されていないため、認証が必要です。
403	禁止されている	認証エラーによりアクセスが拒否されました。
404	が見つかりません	要求で参照されているリソースが存在しません。
409	競合しています	オブジェクトがすでに存在するため、オブジェクトの作成に失敗しました。
500	内部エラー	サーバで一般的な内部エラーが発生しました。
503	サービスを利用できません	サービスは何らかの理由で要求を処理する準備ができていません。

URL 形式

REST API を使用したリソースインスタンスまたはコレクションへのアクセスに使用される URL の一般的な構造は、いくつかの値で構成されます。この構造は、基礎となるオブジェクトモデルとシステム設計を反映しています。

root としてアカウントを作成します

すべての REST エンドポイントへのリソースパスのルートは Astra アカウントです。そして、 URL 内のすべてのパスは「 /account/{account_id} 」で始まります。ここで、「 account_id 」はアカウントの一意の UUIDv4 値です。内部構造：すべてのリソースアクセスが特定のアカウントに基づいている設計を反映します。

エンドポイントリソースカテゴリ

Astra リソースエンドポイントは、次の 3 つのカテゴリに分類されます。

- コア (`/core')
- 管理対象アプリケーション (`/k8s')
- トポロジ (`/topology')

を参照してください "リソース" を参照してください。

カテゴリバージョン

3 つのリソースカテゴリのそれぞれに、アクセスされるリソースのバージョンを制御するグローバルバージョンがあります。規則と定義により ' リソース・カテゴリの新しいメジャー・バージョン (/v1' から /v2') に移行すると 'API の変更が破壊されます

リソースインスタンスまたはコレクション

リソースインスタンスまたはコレクションにアクセスするかどうかに基づいて、パスでリソースタイプと識別子の組み合わせを使用できます。

例

- リソースパス

  上記の構造に基づいて、エンドポイントへの一般的なパスは、「 /accounts/{account_id}/core/v1/users` です。

- 完全な URL

  対応するエンドポイントの完全な URL は、 https://astra.netapp.io/accounts/{account_id}/core/v1/users` です。

リソースとエンドポイント

Astra Control REST API のリソースを使用すると、 Astra の導入を自動化できます。各リソースは、 1 つ以上のエンドポイントを介してアクセスされます。以下に、自動化導入の一環として使用できる REST リソースの概要を示します。

> Astra Control リソースへのアクセスに使用されるパスと完全な URL の形式は、いくつかの値に基づいています。を参照してください "URL 形式" を参照してください。も参照してください "API リファレンス" Astra のリソースとエンドポイントの使用方法の詳細については、を参照してください。

Astra Control REST リソースの要約

Astra Control REST API に用意されているプライマリリソースエンドポイントは、 3 つのカテゴリに分類されています。各リソースは、特定の場所を除いて、すべての CRUD 操作（作成、読み取り、更新、削除）を使用してアクセスできます。

[* リリース *] 列は、リソースが最初に導入されたときのアストラリリースを示します。このフィールドは、現在のリリースで新しく追加されたリソースに対して太字で表示されます。

コアリソース

コアリソースエンドポイントは、 Astra ランタイム環境の確立と保守に必要な基盤サービスを提供します。

リソース	リリース。	説明
アカウント：	21.12	アカウントリソースを使用すると、マルチテナント Astra Control 導入環境内の分離されたテナントを管理できます。
ASUP	21.08	ASUP リソースには、ネットアップサポートに転送される AutoSupport バンドルが表示されます。
クレデンシャル	21.04	クレデンシャルリソースには、 Astra ユーザ、クラスタ、バケット、ストレージバックエンドで使用できるセキュリティ関連の情報が含まれています。
使用権	21.08	使用権リソースは、アクティブなライセンスとサブスクリプションに基づいて、アカウントで使用可能な機能と容量を表します。
イベント	21.04	イベントリソースは、通知として分類されたサブセットも含めて、システムで発生するすべてのイベントを表します。
実行フック	21.12	実行フックリソースは、管理対象アプリケーションのスナップショットの実行前または実行後に実行できるカスタムスクリプトを表します。
フィーチャー（ Feature ）	21.08	機能リソースは、選択した Astra 機能を表します。この機能を照会することで、システムで有効になっているか無効になっているかを確認できます。アクセスは読み取り専用に制限されます。
フックソース	21.12	フックソースリソースは、実行フックで使用される実際のソースコードを表します。ソースコードを実行コントロールから分離すると、スクリプトを共有できるようにするなど、いくつかの利点があります。

リソース	リリース。	説明
使用許諾	21.08	ライセンスリソースは、 Astra アカウントで使用可能なライセンスを表します。
通知	21.04	通知リソースは、通知の送信先を持つ Astra イベントを表します。アクセスはユーザ単位で提供されます。
パッケージ	*22.04 *	パッケージリソースは、パッケージ定義の登録とアクセスを提供します。ソフトウェアパッケージは、ファイル、イメージ、その他のアーティファクトなどのさまざまなコンポーネントで構成されます。
ロールのバインド	21.04	ロールバインドリソースは ' 特定のユーザーとアカウントのペア間の関係を表します2 つの間のリンクに加えて、特定のロールを通じて各に一連の権限が指定されます。
設定	21.08	設定リソースは、特定のアストラアカウントの機能を説明するキーと値のペアの集まりです。
サブスクリプション。	21.08	サブスクリプションリソースは、 Astra アカウントのアクティブなサブスクリプションです。
トークン	21.04	トークンリソースとは、プログラムによって Astra Control REST API にアクセスできるトークンのことです。
未読通知	21.04	未読通知リソースは、特定のユーザーに割り当てられているがまだ読み込まれていない通知を表します。
アップグレード	*22.04 *	アップグレードリソースを使用して、ソフトウェアコンポーネントにアクセスしたり、アップグレードを開始したりできます。
ユーザ	21.04	ユーザリソースは、 Astra を使用するユーザで、定義したロールに基づいてシステムにアクセスできる。

アプリケーションリソースの管理

管理対象アプリケーションリソースエンドポイントは、管理対象の Kubernetes アプリケーションへのアクセスを提供します。

リソース	リリース。	説明
アプリケーションアセット	21.04	アプリケーションアセットリソースは、 Astra アプリケーションの管理に必要な状態情報の内部コレクションです。
アプリケーションのバックアップ	21.04	アプリケーションのバックアップリソースは、管理対象アプリケーションのバックアップを表します。
アプリケーションスナップショット	21.04	アプリケーションスナップショットリソースは、管理対象アプリケーションのスナップショットを表します。
実行フックのオーバーライド	21.12	実行フックの上書きリソースを使用すると、特定のアプリケーションにプリインストールされているネットアップのデフォルトの実行フックを必要に応じて無効にできます。
マネージドアプリケーション	21.04	マネージドアプリケーションリソースは、 Astra が管理する Kubernetes アプリケーションを表しています。

リソース	リリース。	説明
スケジュール	21.04	スケジュールリソースとは、データ保護ポリシーの一部として管理対象アプリケーションにスケジュールされているデータ保護処理のことです。

トポロジリソース

トポロジリソースエンドポイントは、管理対象外のアプリケーションとストレージリソースへのアクセスを提供します。

リソース	リリース。	説明
アプリケーション	21.04	アプリケーションリソースは、 Astra が管理していないアプリケーションも含め、 Kubernetes のすべてのアプリケーションを表します。
バケット	21.08	バケットリソースは、 Astra が管理するアプリケーションのバックアップを保存するために使用する S3 クラウドバケットです。
クラウド	21.08	クラウドリソースとは、アストラクライアントから接続してクラスタやアプリケーションを管理できるクラウドのことです。
クラスタ	21.08	クラスタリソースは Kubernetes で管理されない Kubernetes クラスタを表します。
クラスタノード	21.12	クラスタノードリソースは、 Kubernetes クラスタ内の個々のノードにアクセスできるようにすることで、解決策を提供します。
管理対象クラスタ	21.08	管理対象クラスタリソースは、 Kubernetes で現在管理されている Kubernetes クラスタを表します。
管理対象のストレージバックエンド	21.12	管理対象のストレージバックエンドリソースを使用して、バックエンドストレージプロバイダの抽象化された表現にアクセスできます。これらのストレージバックエンドは、管理対象のクラスタやアプリケーションで使用できます。
ネームスペース	21.12	ネームスペースリソースは、 Kubernetes クラスタ内で使用されるネームスペースへのアクセスを提供します。
ストレージバックエンド	21.08	ストレージバックエンドリソースは、 Astra が管理するクラスタとアプリケーションで使用できるストレージサービスのプロバイダです。
ストレージクラス	21.08	ストレージクラスのリソースは、さまざまなクラスやタイプのストレージを表しており、特定の管理対象クラスタで使用できます。
ボリューム	21.04	ボリュームリソースは、管理対象アプリケーションに関連付けられた Kubernetes ストレージボリュームを表します。

現在のリリースの新しいエンドポイント

現行の 22.04 Astra Control リリースでは、次の REST エンドポイントが追加されています。また、いくつかの既存リソースのバージョンもアップグレードされています。

- /accounts / ｛ account_id ｝ /core/v1/packages
- /accounts / ｛ account_id ｝ /core/v1/packages/ ｛ package_id ｝

- /accounts / { account_id } /core/v1/upgrades
- /accounts / { account_id } /core/v1/upgrades/ { upgrade_id }
- /accounts/{account_id}/topology/v1/appBackups を指定します
- /accounts / { account_id } /topology/v1/appBackups / { appBackup_id }
- /accounts / { account_id } /topology/v1/clouds/ { cloud_id } /clusters/ { cluster_id } /clusterNodes
- /accounts / { account_id } /topology/v1/clouds/ { cloud_id } /clusters/ { cluster_id } /clusterNodes / { clusternode_id }
- /accounts / { account_id } /topology/v1/managedClusters / { managedCluster_id } /apps/ { app_id } /appAssets
- /accounts / { account_id } /topology/v1/managedClusters / { managedCluster_id } /apps/ { APP_id } /appAssets/ { appAsset_id }
- /accounts / { account_id } /topology/v1/managedClusters / { managedCluster_id } / clusterNodes
- /accounts / { account_id } /topology/v1/managedClusters / { managedCluster_id } /clusterNodes / { clusternode_id }

その他のリソースとエンドポイント

Astra の導入をサポートするために使用できる追加のリソースとエンドポイントがいくつかあります。

これらのリソースとエンドポイントは、現在のところ、 Astra Control REST API リファレンスドキュメントに含まれていません。

OpenAPI

OpenAPI エンドポイントは、現在の OpenAPI JSON ドキュメントおよびその他の関連リソースへのアクセスを提供します。

OpenMetrics

OpenMetrics エンドポイントは、 OpenMetrics リソースを介してアカウントメトリックへのアクセスを提供します。サポートは、 Astra Control Center 導入モデルで利用できます。

その他の使用に関する考慮事項

RBAC セキュリティ

Astra REST API は、ロールベースアクセス制御（ RBAC ）をサポートしているため、システム機能へのアクセスを制限できます。

Astra の役割

すべての Astra ユーザには、実行可能なアクションを決定する 1 つのロールが割り当てられます。役割は、次の表に示すように階層構造になっています。

ロール	説明
オーナー	admin ロールのすべての権限が割り当てられており、 Astra アカウントを削除することもできます。
管理	メンバーの役割のすべての権限を持ち、ユーザーをアカウントに招待することもできます。
メンバー	Astra アプリケーションとコンピューティングのリソースを完全に管理できる。
ビューアー（ Viewer ）	リソースの表示のみに制限されます。

ネームスペース単位で強化された RBAC

この機能は、 Astra REST API の 22.04 リリースで導入されました。

特定のユーザに対してロールバインドが確立されている場合は、制約を適用して、ユーザがアクセスできるネームスペースを制限できます。次の表に示すように、この制約を定義する方法はいくつかあります。詳細については ' ロールバインド API の roleContraints パラメータを参照してください

ネームスペース	説明
すべて	ユーザは、ワイルドカードパラメータ「 * 」を使用してすべてのネームスペースにアクセスできます。これは、下位互換性を維持するためのデフォルト値です。
なし	拘束リストは指定されますが、空です。これは、ユーザがどのネームスペースにもアクセスできないことを示します。
ネームスペースリスト	ユーザを 1 つのネームスペースに制限するネームスペースの UUID が含まれています。カンマで区切ったリストを使用して、複数のネームスペースへのアクセスを許可することもできます。
ラベル	ラベルが指定され、一致するすべてのネームスペースへのアクセスが許可されます。

コレクションを操作する

Astra Control REST API には、定義されたクエリパラメータを使用してリソースコレクションにアクセスするためのさまざまな方法があります。

値の選択

各リソース・インスタンスに対して 'include' パラメータを使用して ' どのキーと値のペアを返すかを指定できますすべてのインスタンスが応答の本文で返されます。

フィルタリング

収集リソースのフィルタリングを使用すると、 API ユーザは、応答の本文でリソースが返されるかどうかを決定する条件を指定できます。「 filter 」パラメータは、フィルタリング条件を示すために使用されます。

並べ替え

収集リソースのソートを使用すると、 API ユーザは応答の本文でリソースが返される順序を指定できます。「 orderBy 」パラメータは、フィルタリング条件を示します。

ページ付け

制限パラメータを使用して ' 要求に対して返されるリソース・インスタンスの数を制限することにより ' ページ付けを強制できます

カウント

ブール値パラメータ「 count 」を「 true 」に設定した場合、返された応答の配列内のリソースの数がメタデータセクションに表示されます。

診断とサポート

診断とデバッグに使用できる Astra Control REST API には、いくつかのサポート機能が用意されています。

API リソース

API リソースからは、診断情報とサポート情報を提供する Astra 機能がいくつか提供されています。

を入力します	説明
イベント	アストラ処理の一部として記録されるシステムアクティビティ。
通知	ユーザに提供するのに十分な重要性があると見なされるイベントのサブセット。
未読通知	ユーザがまだ読み取りまたは取得していない通知。

API トークンを取り消します

不要になった API トークンは、 Astra Web インターフェイスで取り消すことができます。

作業を開始する前に

Astra アカウントが必要。また、取り消すトークンを特定する必要があります。

このタスクについて

トークンが取り消されると、そのトークンはただちに永続的に使用できなくなります。

手順

1. アカウントのクレデンシャルを使用して Astra にサインインします。

   Astra Control Service の次のサイトにアクセスします。 "https://astra.netapp.io"

2. ページの右上にある図のアイコンをクリックし、 * API access * を選択します。
3. 取り消すトークンまたはトークンを選択します。
4. [* アクション * （ * Actions * ）] ドロップダウンボックスで、 [トークンの無効化 * （ * Revoke tokens * ）] をクリック

インフラワークフロー

作業を開始する前に

これらのワークフローを使用して、 Astra Control Center 導入モデルで使用されるインフラストラクチャを作成および管理できます。ほとんどの場合、 Astra Control Service でワークフローを使用することもできます。

> これらのワークフローは、ネットアップがいつでも拡張および強化できるため、定期的に確認する必要があります。

一般的な準備

いずれかの Astra ワークフローを使用する前に、を確認してください "ワークフローを使用する準備をします"。

ワークフローのカテゴリ

インフラのワークフローは、さまざまなカテゴリに分類されているため、必要なワークフローを簡単に見つけることができます。

カテゴリ	説明
ID とアクセス	これらのワークフローを使用すると、 Astra のアイデンティティとアクセス方法を管理できます。リソースには、ユーザ、クレデンシャル、トークンが含まれます。
バケット	これらのワークフローを使用して、バックアップの格納に使用する S3 バケットを作成および管理できます。
ストレージ	これらのワークフローを使用して、ストレージバックエンドとストレージボリュームを追加および管理できます。
クラスタ	管理対象の Kubernetes クラスタを追加すると、クラスタに含まれるアプリケーションを保護およびサポートできるようになります。

ID とアクセス

ユーザをリストします

特定のアストラアカウントに対して定義されているユーザをリストできます。

1. ユーザをリストします

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
取得	/account/{accountID}/core/v1/users

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
含める	クエリ	いいえ	必要に応じて、応答で返す値を選択します。

curl の例：すべてのユーザのすべてのデータを返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/core/v1/users' --header
'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

curl の例：すべてのユーザの名前、姓、および **ID** を返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/core/v1/users?include=first
Name,lastName,id' --header 'Accept: */*' --header 'Authorization: Bearer
<API_TOKEN>'
```

JSON 出力例

```
{
  "items": [
    [
    "David",
    "Peterson",
    "844ec6234-11e0-49ea-8434-a992a6270ec1"
    ],
    [
      "Scott",
      "Morris",
      "2a3e227c-fda7-4145-a86c-ed9aa0183a6c"
    ]
  ],
  "metadata": {}
}
```

バケット

バケットをリストします

特定のアストラアカウント用に定義された S3 バケットをリストできます。

1. バケットをリストします

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
取得	/account/{accountID}/topology/v1/buckets/

curl の例：すべてのバケットのすべてのデータを返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/topology/v1/buckets'
--header 'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

ストレージ

ストレージバックエンドをリストします

使用可能なストレージバックエンドを表示できます。

1. バケットをリストします

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
取得	/account/{accountID}/topology/v1/storageBackends

cURL の例：すべてのストレージバックエンドのすべてのデータを返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/topology/v1/storageBackends
' --header 'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

JSON 出力例

```
{
  "items": [
    {
      "backendCredentialsName": "10.191.77.177",
      "backendName": "myinchunhcluster-1",
      "backendType": "ONTAP",
      "backendVersion": "9.8.0",
      "configVersion": "Not applicable",
      "health": "Not applicable",
      "id": "46467c16-1585-4b71-8e7f-f0bc5ff9da15",
      "location": "nalab2",
      "metadata": {
        "createdBy": "4c483a7e-207b-4f9a-87b7-799a4629d7c8",
        "creationTimestamp": "2021-07-30T14:26:19Z",
        "modificationTimestamp": "2021-07-30T14:26:19Z"
      },
      "ontap": {
        "backendManagementIP": "10.191.77.177",
        "managementIPs": [
          "10.191.77.177",
          "10.191.77.179"
        ]
      },
      "protectionPolicy": "Not applicable",
      "region": "Not applicable",
      "state": "Running",
      "stateUnready": [],
      "type": "application/astra-storageBackend",
      "version": "1.0",
      "zone": "Not applicable"
    }
  ]
}
```


クラスタ

管理対象クラスタをリストします

Astra が現在管理している Kubernetes クラスタをリストできます。

1. クラスタを表示します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
取得	/account/{accountID}/topology/v1/managedClusters

curl の例：すべてのクラスタのすべてのデータを返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/topology/v1/managedClusters
' --header 'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

管理ワークフロー

作業を開始する前に

これらのワークフローは、 Astra 管理対象クラスタ内でアプリケーションを管理する際に使用できます。

> これらのワークフローは、ネットアップがいつでも拡張および強化できるため、定期的に確認する必要があります。

一般的な準備

いずれかの Astra ワークフローを使用する前に、を確認してください "ワークフローを使用する準備をします"。

ワークフローのカテゴリ

管理ワークフローは、目的のワークフローを見つけやすくするために、さまざまなカテゴリに分類されています。

カテゴリ	説明
アプリケーション制御	これらのワークフローでは、管理対象アプリケーションと管理対象外アプリケーションを制御できます。アプリをリスト表示したり、管理アプリを作成および削除したりできます。
アプリケーションの保護	これらのワークフローを使用して、 Snapshot とバックアップを通じて管理対象アプリケーションを保護することができます。
アプリケーションのクローニングとリストア	ここでは、管理対象アプリケーションのクローニングとリストアのワークフローについて説明します。
サポート	アプリケーションと一般的な Kubernetes 環境をデバッグおよびサポートするためのワークフローがいくつかあります。

その他の考慮事項については

管理ワークフローを使用する場合は、さらにいくつかの点を考慮する必要があります。

アプリケーションのクローンを作成します

アプリケーションのクローニングでは、いくつかの点を考慮する必要があります。以下に、 JSON 入力のパラメータを示します。

ソースクラスタの識別子

「 ourceClusterID 」の値は、元のアプリケーションがインストールされているクラスタを常に識別します。

クラスタ **ID**

「 ClusterId 」の値は、新しいアプリケーションをインストールするクラスタを識別します。

- 同じクラスタ内でクローニングする場合、「 ClusterId 」と「 ourceClusterID 」には同じ値があります。
- クラスタ間でクローンを作成する場合 ' 2 つの値は異なる値であり 'ClusterId' はターゲット・クラスタの ID である必要があります

ネームスペース

「 namespace 」の値は、元のソースアプリケーションとは異なる値にする必要があります。さらに、クローンのネームスペースを作成できず、 Astra がそのクローンを作成します。

バックアップとスナップショット

オプションで 'backupid' または 'napshotID' パラメータを使用して ' 既存のバックアップまたはスナップショットからアプリケーションのクローンを作成できますバックアップまたはスナップショットを指定しない場合は、まずアプリケーションのバックアップを作成し、次にバックアップからクローンを作成します。

アプリケーションの復元

アプリケーションをリストアする際に考慮すべき事項をいくつか示します。

- アプリケーションのリストアは、クローニング処理とほぼ同じです。
- アプリケーションをリストアするときは、バックアップまたはスナップショットを指定する必要があります。

アプリコントロール

アンマネージアプリケーションをリストします

Astra で現在管理されていないアプリケーションをリストできます。これは、管理対象のアプリケーションを選択する際に行うことができます。

> これらのワークフローで使用されている REST エンドポイントは、すべての Astra アプリケーションをデフォルトで返します。API 呼び出しで「 filter 」クエリーパラメータを使用して、アンマネージアプリケーションのみを返すように要求できます。代わりに、フィルタパラメータを省略してすべてのアプリケーションを返し、出力の「 anagedState 」フィールドを調べて、「 unmanaged 」状態のアプリケーションを判別することもできます。

managedState が **unmanaged** のアプリケーションのみをリストします

このワークフローでは 'filter' クエリー・パラメータを使用して ' 管理対象外のアプリケーションのみを返します

1. 管理対象外のアプリケーションを一覧表示します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
取得	/account/{accountID}/topology/v1/apps の順で指定します

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
フィルタ	クエリ	いいえ	フィルタを使用して、どのアプリケーションを返すかを指定します。
含める	クエリ	いいえ	必要に応じて、応答で返す値を選択します。

curl の例：アンマネージアプリケーションの名前、 **ID** 、および **managedState** を返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/topology/v1/apps?filter=man
agedState%20eq%20'unmanaged'&include=name,id,managedState' --header
'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

JSON 出力例

```
{
  "items": [
    [
      "maria",
      "eed19f78-0884-4792-bb7a-313258c6b0b1",
      "unmanaged"
    ],
    [
      "test-postgres-app",
      "1ee6235b-cda1-45cb-8d4c-630bdb8b41a5",
      "unmanaged"
    ],
    [
      "postgres1-postgresql",
      "e591ee59-ea90-4a9f-8e6c-d2b6e8647096",
      "unmanaged"
    ],
    [
      "kube-system",
      "077a2f73-4b51-4d04-8c6c-f63b3b069755",
      "unmanaged"
    ],
    [
      "trident",
      "5b6fc28f-e308-4653-b9d2-6d66a764d2e1",
      "unmanaged"
    ],
    [
      "postgres1-postgresql-clone",
      "06be05c5-763e-4d73-bd06-1f27f5f2e130",
      "unmanaged"
    ]
  ],
  "metadata": {}
}
```


すべてのアプリケーションをリストし、管理対象外のアプリケーションを選択します

このワークフローはすべてのアプリケーションを返します。出力を調べて、どちらが管理対象外であるかを確認する必要があります。

1. すべてのアプリケーションを一覧表示します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
取得	/account/{accountID}/topology/v1/apps の順で指定します

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
含める	クエリ	いいえ	必要に応じて、応答で返す値を選択します。

curl の例：すべてのアプリケーションのすべてのデータを返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/topology/v1/apps' --header
'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

curl の例：すべてのアプリケーションの名前、 **ID** 、 **managedState** を返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/topology/v1/apps?include=na
me,id,managedState' --header 'Accept: */*' --header 'Authorization: Bearer
<API_TOKEN>'
```

JSON 出力例

```
{
  "items": [
    [
      "maria",
      "eed19f78-0884-4792-bb7a-313258c6b0b1",
      "unmanaged"
    ],
    [
      "mariadb-mariadb",
      "8da20fff-c69c-4170-bb0d-e4f91c5a1333",
      "managed"
    ],
    [
      "test-postgres-app",
      "1ee6235b-cda1-45cb-8d4c-630bdb8b41a5",
      "unmanaged"
    ],
    [
      "postgres1-postgresql",
      "e591ee59-ea90-4a9f-8e6c-d2b6e8647096",
      "unmanaged"
    ],
    [
      "kube-system",
      "077a2f73-4b51-4d04-8c6c-f63b3b069755",
      "unmanaged"
    ],
    [
      "trident",
      "5b6fc28f-e308-4653-b9d2-6d66a764d2e1",
      "unmanaged"
    ],
    [
      "postgres1-postgresql-clone",
      "06be05c5-763e-4d73-bd06-1f27f5f2e130",
      "unmanaged"
    ],
    [
      "davidns-postgres-app",
      "11e046b7-ec64-4184-85b3-debcc3b1da4d",
      "managed"
    ]
  ],
  "metadata": {}
}
```

2. 管理されていないアプリケーションを選択します

API 呼び出しの出力を確認し、「 unmanaged 」に等しい「 anagedState 」を持つアプリケーションを手動で選択します。

管理対象アプリケーションをリストします

Astra が現在管理しているアプリケーションをリストできます。特定のアプリケーションの Snapshot やバックアップを検索する際に、この操作を行うことができます。

1. アプリケーションを一覧表示します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
取得	/account/{accountID}/k8s/v1/managedApps

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
含める	クエリ	いいえ	必要に応じて、応答で返す値を選択します。

curl の例：すべてのアプリケーションのすべてのデータを返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps'
--header 'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

curl の例：すべてのアプリケーションの名前、 **ID** 、および状態を返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps?include=
name,id,state' --header 'Accept: */*' --header 'Authorization: Bearer
<API_TOKEN>'
```

JSON 出力例

```
{
  "items": [
    [
      "test-postgres-app",
      "1ee6235b-cda1-45cb-8d4c-630bdb8b41a5",
      "running"
    ]
  ],
  "metadata": {}
}
```

管理アプリを取得します

単一の管理対象アプリケーションを記述するすべてのリソース変数を取得できます。

作業を開始する前に

取得する管理アプリの ID が必要です。必要に応じて、ワークフローを使用できます "管理対象アプリケーションをリストします" アプリケーションを検索します。

1. アプリケーションを取得します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
取得	/accounts / { account_id } /k8s/v1/managedApps/ { managedApp_id }

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
管理アプリ ID	パス	はい。	取得する管理アプリケーションの ID 値。

curl の例：アプリケーションのすべてのデータを返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps/<MANAGED
_APP_ID>' --header 'Accept: */*' --header 'Authorization: Bearer
<API_TOKEN>'
```

アプリを管理します

Astra がすでに認識しているアプリケーションに基づいて、管理対象アプリケーションを作成できます。アプリケーションの管理時には、定期的なバックアップとスナップショットを作成してアプリケーションを保護できます。

作業を開始する前に

管理する検出されたアプリケーションの ID が必要です。必要に応じて、ワークフローを使用できます "アンマネージアプリケーションをリストします" アプリケーションを検索します。

1. アプリケーションを管理します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
投稿（ Post ）	/account/{accountID}/k8s/v1/managedApps

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
JSON	ボディ（ Body ）	はい。	管理対象アプリケーションを識別するために必要なパラメータを提供します。以下の例を参照してください。

JSON の入力例

```
{
  "type": "application/astra-managedApp",
  "version": "1.1",
  "id": "7da20fff-c69d-4270-bb0d-a4f91c5a1333"
}
```

curl の例：アプリケーションの管理

```
curl --location -i --request POST
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps'
--header 'Content-Type: application/astra-managedApp+json' --header
'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'  --d @JSONinput
```

アプリの管理を解除します

管理対象アプリが不要になった場合は削除できます。管理アプリケーションを削除すると、関連付けられているスケジュールも削除されます。

作業を開始する前に

管理を解除する管理対象アプリの ID が必要です。必要に応じて、ワークフローを使用できます "管理対象アプリケーションをリストします" アプリケーションを検索します。

アプリケーションのバックアップと Snapshot は、削除されても自動的には削除されません。バックアップと Snapshot が不要になった場合は、アプリケーションを削除する前に削除する必要があります。

1. アプリをアンマネージします

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
削除	/accounts / ｛ account_id ｝ /k8s/v1/managedApps/ ｛ managedApp_id ｝

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
管理アプリ ID	パス	はい。	削除する管理対象アプリケーションを指定します。

curl の例：管理対象アプリケーションを削除します

```
curl --location -i --request DELETE
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps/<MANAGED
_APP_ID>' --header 'Accept: */*' --header 'Authorization: Bearer
<API_TOKEN>'
```

アプリケーションの保護

Snapshot を一覧表示します

特定の管理対象アプリケーションに対して作成されたスナップショットを一覧表示できます。

作業を開始する前に

スナップショットを表示する管理対象アプリケーションの ID が必要です。必要に応じて、ワークフローを使

用できます "管理対象アプリケーションをリストします" アプリケーションを検索します。

1. スナップショットを一覧表示します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
取得	/accounts / ｛ account_id ｝ /k8s/v1/managedApps/ ｛ managedApp_id ｝ /appSnaps

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
管理アプリ ID	パス	はい。	リストされたスナップショットを所有する管理アプリケーションを識別します。
カウント	クエリ	いいえ	'count=true' の場合 ' スナップショットの数は応答のメタデータセクションに含まれます

curl の例：アプリケーションのすべてのスナップショットを返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps/<MANAGED
_APP_ID>/appSnaps' --header 'Accept: */*' --header 'Authorization: Bearer
<API_TOKEN>'
```

curl の例：アプリケーションのすべてのスナップショットとカウントを返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps/<MANAGED
_APP_ID>/appSnaps?count=true' --header 'Accept: */*' --header
'Authorization: Bearer <API_TOKEN>'
```

JSON 出力例

```
{
  "items": [
    {
      "id": "dc2974ae-f71d-4c81-91b5-f96cf72dc3ba",
      "metadata": {
        "createdBy": "fb093413-b6fc-4a64-a48a-afc32ada8537",
        "creationTimestamp": "2021-06-04T21:23:14Z",
        "modificationTimestamp": "2021-06-04T21:23:14Z",
        "labels": []
      },
      "snapshotAppAsset": "4547658d-cc06-4c1d-ad8a-4a05274d0db0",
      "snapshotCreationTimestamp": "2021-06-04T21:23:47Z",
      "name": "test-postgres-app-snapshot-20210604212213",
      "state": "completed",
      "stateUnready": [],
      "type": "application/astra-appSnap",
      "version": "1.0"
    }
  ],
  "metadata": {
    "count": 1
  }
}
```


バックアップをリスト表示します

特定の管理対象アプリケーション用に作成されたバックアップをリストできます。

作業を開始する前に

バックアップを表示する管理対象アプリケーションの ID が必要です。必要に応じて、ワークフローを使用できます "管理対象アプリケーションをリストします" アプリケーションを検索します。

1. バックアップを一覧表示します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
取得	/accounts / { account_id } /k8s/v1/managedApps/ { managedApp_id } /appBackups

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
管理アプリ ID	パス	はい。	リストされたバックアップを所有する管理アプリケーションを指定します。

curl の例：アプリケーションのすべてのバックアップを返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps/<MANAGED
_APP_ID>/appBackups' --header 'Accept: */*' --header 'Authorization:
Bearer <API_TOKEN>'
```

JSON 出力例

```
{
  "items": [
    {
      "type": "application/astra-appBackup",
      "version": "1.0",
      "id": "ed39fdb0-12db-497b-9e46-20036c1fb0d2",
      "name": "mariadb-mariadb-backup-20210617175900",
      "state": "completed",
      "stateUnready": [],
      "bytesDone": 0,
      "percentDone": 100,
      "metadata": {
        "labels": [],
        "creationTimestamp": "2021-06-17T17:59:09Z",
        "modificationTimestamp": "2021-06-17T17:59:09Z",
        "createdBy": "fb093413-b6fc-4a64-a48a-afc32ada8537"
      }
    }
  ],
  "metadata": {}
}
```

管理アプリのスナップショットを作成します

特定の管理対象アプリケーションのスナップショットを作成できます。

作業を開始する前に

スナップショットを作成する管理対象アプリケーションの ID が必要です。必要に応じて、ワークフローを使用できます "管理対象アプリケーションをリストします" アプリケーションを検索します。

1. スナップショットを作成します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
投稿（ Post ）	/accounts / ｛ account_id ｝ /k8s/v1/managedApps/ ｛ managedApp_id ｝ /appSnaps

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
管理アプリ ID	パス	はい。	Snapshot を作成する管理対象アプリケーションを指定します。
JSON	ボディ（ Body ）	はい。	Snapshot のパラメータを提供します。以下の例を参照してください。

JSON の入力例

```
{
  "type": "application/astra-appSnap",
  "version": "1.0",
  "name": "snapshot-david-1"
}
```

curl の例：アプリケーションのスナップショットを作成します

```
curl --location -i --request POST
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps/<MANAGED
_APP_ID>/appSnaps' --header 'Content-Type: application/astra-appSnap+json'
--header 'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>' --d
@JSONinput
```

管理対象アプリケーションのバックアップを作成

特定の管理対象アプリケーションのバックアップを作成できます。バックアップを使用して、アプリケーションのリストアやクローニングを行うことができます。

作業を開始する前に

バックアップを作成する管理対象アプリケーションの ID が必要です。必要に応じて、ワークフローを使用できます "管理対象アプリケーションをリストします" アプリケーションを検索します。

1. バックアップを作成します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
投稿（ Post ）	/accounts / ｛ account_id ｝ /k8s/v1/managedApps/ ｛ managedApp_id ｝ /appBackups

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
管理アプリ ID	パス	はい。	バックアップを作成する管理対象アプリケーションを指定します。
JSON	ボディ（ Body ）	はい。	バックアップのパラメータが表示されます。以下の例を参照してください。

JSON の入力例

```
{
  "type": "application/astra-appBackup",
  "version": "1.0",
  "name": "backup-david-1"
}
```

curl の例：アプリケーションのバックアップを作成します

```
curl --location -i --request POST
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps/<MANAGED
_APP_ID>/appBackups' --header 'Content-Type: application/astra-
appBackup+json' --header 'Accept: */*' --header 'Authorization: Bearer
<API_TOKEN>' --d @JSONinput
```

Snapshot を削除します

管理対象アプリケーションに関連付けられている Snapshot を削除することができます。

作業を開始する前に

次の情報が必要です。

- スナップショットを所有する管理対象アプリケーションの ID 。必要に応じて、ワークフローを使用できます "管理対象アプリケーションをリストします" アプリケーションを検索します。
- 削除する Snapshot の ID 。必要に応じて、ワークフローを使用できます "Snapshot を一覧表示します" をクリックしてください。

1. スナップショットを削除します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
削除	/accounts / { account_id } /k8s/v1/managedApps_ { managedApp_id } /appSnaps/ { appSnap_id }

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
管理アプリ ID	パス	はい。	スナップショットを所有する管理アプリケーションを指定します。
Snapshot ID	パス	はい。	削除する Snapshot を指定します。

curl の例：アプリケーションのスナップショットを **1** つ削除します

```
curl --location -i --request DELETE
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps/<MANAGED
_APP_ID>/appSnaps/<SNAPSHOT_ID>' --header 'Accept: */*' --header
'Authorization: Bearer <API_TOKEN>'
```

バックアップを削除します

管理対象アプリケーションに関連付けられているバックアップを削除することができます。

作業を開始する前に

次の情報が必要です。

- バックアップを所有する管理対象アプリケーションの ID 。必要に応じて、ワークフローを使用できます "管理対象アプリケーションをリストします" アプリケーションを検索します。
- 削除するバックアップの ID 。必要に応じて、ワークフローを使用できます "バックアップをリスト表示します" をクリックしてください。

1. バックアップを削除します

次の REST API 呼び出しを実行します。

> 障害が発生したバックアップは、次に示すオプションの要求ヘッダーを使用して強制的に削除できます。

HTTP メソッド	パス
削除	/accounts / { account_id } /k8s/v1/managedApps / { managedApp_id } /appBackups / { appBackup_id }

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
管理アプリ ID	パス	はい。	バックアップを所有する管理アプリケーションを指定します。
バックアップ ID	パス	はい。	削除するバックアップを指定します。
強制的に削除します	ヘッダー	いいえ	失敗したバックアップを強制的に削除する場合に使用します。

curl の例：アプリケーションのバックアップを **1** つ削除します

```
curl --location -i --request DELETE
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps/<MANAGED
_APP_ID>/appBackups/<BACKUP_ID>' --header 'Accept: */*' --header
'Authorization: Bearer <API_TOKEN>'
```

curl の例： **force** オプションを使用して、アプリケーションのバックアップを **1** つ削除します

```
curl --location -i --request DELETE
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps/<MANAGED
_APP_ID>/appBackups/<BACKUP_ID>' --header 'Accept: */*' --header
'Authorization: Bearer <API_TOKEN>' --header 'Force-Delete: true'
```

アプリケーションのクローニングとリストア

管理対象アプリケーションのクローンを作成します

既存の管理対象アプリケーションをクローニングして、新しいアプリケーションを作成できます。

作業を開始する前に

このワークフローについては、次の点に注意してください。

- アプリケーションのバックアップまたはスナップショットは使用されません
- クローニング処理は同じクラスタ内で実行されます

> アプリケーションを別のクラスタにクローニングするには、 JSON 入力の「 clusterId 」パラメータを環境に応じて更新する必要があります。

1. クローンを作成する管理アプリを選択します

ワークフローを実行 "管理対象アプリケーションをリストします" をクリックし、クローニングするアプリケーションを選択します。アプリケーションのクローニングに使用される REST 呼び出しには、いくつかのリソース値が必要です。

2. アプリを複製します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
投稿（ Post ）	/account/{accountID}/k8s/v1/managedApps

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
JSON	ボディ（ Body ）	はい。	クローニングされたアプリケーションのパラメーターを提供します。以下の例を参照してください。

JSON の入力例

```
{
  "type": "application/astra-managedApp",
  "version": "1.0",
  "name": "postgres1-postgresql-clone",
  "clusterID": "30880586-d579-4d27-930f-a9633e59173b",
  "sourceClusterID": "30880586-d579-4d27-930f-a9633e59173b",
  "namespace": "davidns-postgres-app",
  "sourceAppID": "e591ee59-ea90-4a9f-8e6c-d2b6e8647096"
}
```

curl の例：アプリケーションのクローンを作成します

```
curl --location -i --request POST
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps'
--header 'Content-Type: application/astra-managedApp+json' --header '*/*'
--header 'Authorization: Bearer <API_TOKEN>' --d @JSONinput
```

スナップショットから管理アプリのクローンを作成します

アプリケーションスナップショットからアプリケーションをクローニングして、新しいアプリケーションを作成できます。

作業を開始する前に

このワークフローについては、次の点に注意してください。

- アプリケーションスナップショットが使用されます
- クローニング処理は同じクラスタ内で実行されます

> アプリケーションを別のクラスタにクローニングするには、 JSON 入力の「 clusterId 」パラメータを環境に応じて更新する必要があります。

1. クローンを作成する管理アプリを選択します

ワークフローを実行 "管理対象アプリケーションをリストします" をクリックし、クローニングするアプリケーションを選択します。アプリケーションのクローニングに使用される REST 呼び出しには、いくつかのリソース値が必要です。

2. 使用するスナップショットを選択します

ワークフローを実行 "Snapshot を一覧表示します" をクリックし、使用する Snapshot を選択します。

3. アプリを複製します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
投稿（ Post ）	/account/{accountID}/k8s/v1/managedApps

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
JSON	ボディ（Body）	はい。	クローニングされたアプリケーションのパラメーターを提供します。以下の例を参照してください。

JSON の入力例

```
{
  "type": "application/astra-managedApp",
  "version": "1.0",
  "name": "postgres1-postgresql-clone",
  "clusterID": "30880586-d579-4d27-930f-a9633e59173b",
  "sourceClusterID": "30880586-d579-4d27-930f-a9633e59173b",
  "namespace": "davidns-postgres-app",
  "snapshotID": "e24515bd-a28e-4b28-b832-f3c74dbf32fb",
  "sourceAppID": "e591ee59-ea90-4a9f-8e6c-d2b6e8647096"
}
```

curl の例： **Snapshot** からアプリケーションをクローニングします

```
curl --location -i --request POST
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps'
--header 'Content-Type: application/astra-managedApp+json' --header '*/*'
--header 'Authorization: Bearer <API_TOKEN>' --d @JSONinput
```

バックアップから管理対象アプリケーションをクローニングする

アプリケーションバックアップからクローニングすることで、新しい管理対象アプリケーションを作成できます。

作業を開始する前に

このワークフローについては、次の点に注意してください。

- アプリケーションのバックアップが使用されます
- クローニング処理は同じクラスタ内で実行されます

> アプリケーションを別のクラスタにクローニングするには、 JSON 入力の「 clusterId 」パラメータを環境に応じて更新する必要があります。

1. クローンを作成する管理アプリを選択します

ワークフローを実行 "管理対象アプリケーションをリストします" をクリックし、クローニングするアプリケーションを選択します。アプリケーションのクローニングに使用される REST 呼び出しには、いくつかのリソース値が必要です。

2. 使用するバックアップを選択します

ワークフローを実行 "バックアップをリスト表示します" 使用するバックアップを選択します。

3. アプリを複製します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
投稿（ Post ）	/account/{accountID}/k8s/v1/managedApps

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
JSON	ボディ（ Body ）	はい。	クローニングされたアプリケーションのパラメーターを提供します。以下の例を参照してください。

JSON の入力例

```
{
  "type": "application/astra-managedApp",
  "version": "1.0",
  "name": "postgres1-postgresql-clone",
  "clusterID": "30880586-d579-4d27-930f-a9633e59173b",
  "sourceClusterID": "30880586-d579-4d27-930f-a9633e59173b",
  "namespace": "davidns-postgres-app",
  "backupID": "e24515bd-a28e-4b28-b832-f3c74dbf32fb",
  "sourceAppID": "e591ee59-ea90-4a9f-8e6c-d2b6e8647096"
}
```

curl の例：バックアップからアプリケーションをクローニングします

```
curl --location -i --request POST
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps'
--header 'Content-Type: application/astra-managedApp+json' --header '*/*'
--header 'Authorization: Bearer <API_TOKEN>' --d @JSONinput
```

管理アプリをバックアップからリストアします

バックアップから新しいアプリケーションを作成して、管理対象アプリケーションをリストアすることができます。

1. 復元する管理アプリを選択します

ワークフローを実行 "管理対象アプリケーションをリストします" をクリックし、クローニングするアプリケーションを選択します。アプリケーションのクローニングに使用される REST 呼び出しには、いくつかのリソース値が必要です。

2. 使用するバックアップを選択します

ワークフローを実行 "バックアップをリスト表示します" 使用するバックアップを選択します。

3. アプリを復元します

次の REST API 呼び出しを実行します。バックアップ（以下を参照）または Snapshot の ID を指定する必要があります。

HTTP メソッド	パス
PUT	/account/{accountID}/k8s/v1/managedApps/{appID}

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
JSON	ボディ（Body）	はい。	クローニングされたアプリケーションのパラメーターを提供します。以下の例を参照してください。

JSON の入力例

```
{
  "type": "application/astra-managedApp",
  "version": "1.2",
  "backupID": "e24515bd-a28e-4b28-b832-f3c74dbf32fb"
}
```

curl の例：バックアップからアプリを所定の場所にリストアします

```
curl --location -i --request PUT
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/k8s/v1/managedApps/<APP_ID>
' --header 'Content-Type: application/astra-managedApp+json' --header
'*/*' --header 'ForceUpdate: true' --header 'Authorization: Bearer
<API_TOKEN>' --d @JSONinput
```

サポート

通知を一覧表示します

特定のアストラアカウントの通知をリストできます。この処理は、システムアクティビティの監視や問題のデバッグの一環として実行できます。

1. 通知を一覧表示します

次の REST API 呼び出しを実行します。

HTTP メソッド	パス
取得	/account/{accountID}/core/v1/notifications

追加の入力パラメータ

すべての REST API 呼び出しに共通するパラメータに加えて、この手順の curl の例では次のパラメータも使用されます。

パラメータ	を入力します	必須	説明
フィルタ	クエリ	いいえ	必要に応じて、応答で返す通知をフィルタリングします。
含める	クエリ	いいえ	必要に応じて、応答で返す値を選択します。

curl の例：すべての通知を返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/core/v1/notifications'
--header 'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

curl の例：重大度が **WARNING** の通知について概要を返します

```
curl --location -i --request GET
'https://astra.netapp.io/accounts/<ACCOUNT_ID>/core/v1/notifications?filte
r=severity%20eq%20'warning'&include=description' --header 'Accept: */*'
--header 'Authorization: Bearer <API_TOKEN>'
```

JSON 出力例

```
{
  "items": [
    [
      "Trident on cluster david-ie-00 has failed or timed out;
installation of the Trident operator failed or is not yet complete;
operator failed to reach an installed state within 300.00 seconds;
container trident-operator not found in operator deployment"
    ],
    [
      "Trident on cluster david-ie-00 has failed or timed out;
installation of the Trident operator failed or is not yet complete;
operator failed to reach an installed state within 300.00 seconds;
container trident-operator not found in operator deployment"
    ]
  ],
  "metadata": {}
}
```


失敗したアプリを削除します

バックアップまたはスナップショットが失敗した場合、管理アプリを削除できないことがあります。この場合、以下に説明するワークフローを使用して、アプリケーションを手動で削除できます。

1. 削除する管理アプリを選択します

ワークフローを実行 "管理対象アプリケーションをリストします" をクリックし、削除するアプリケーションを選択します。

2. アプリケーションの既存のバックアップを一覧表示します

ワークフローを実行 "バックアップをリスト表示します"。

3. すべてのバックアップを削除します

ワークフローを実行して、すべてのアプリケーションバックアップを削除します "バックアップを削除します" リスト内のバックアップごとに、

4. アプリケーションの既存のスナップショットを一覧表示します

ワークフローを実行 "Snapshot を一覧表示します"。

5. すべてのスナップショットを削除します

ワークフローを実行 "Snapshot を削除します" リスト内の各 Snapshot から削除します。

6. アプリケーションを削除します

ワークフローを実行 "アプリの管理を解除します" アプリケーションを削除します。

Python を使用する

NetApp Astra Control Python SDK

NetApp Astra Control Python SDK は、 Astra Control の導入を自動化するために使用できるオープンソースパッケージです。また、このパッケージは、独自の自動化プラットフォームを作成する際に、おそらく Astra Control REST API について学ぶ際の貴重な資料となります。

> 簡単にするために、このページの以降の部分では、 NetApp Astra Control Python SDK を * SDK と呼びます。

関連する 2 つのソフトウェアツール

SDK には、 Astra Control REST API にアクセスするときに異なる抽象化レベルで動作する 2 つの異なる関連ツールが含まれています。

Astra SDK

Astra SDK は、コアプラットフォーム機能を提供します。基盤となる REST API 呼び出しを抽象化する一連の Python クラスが含まれています。このクラスは、アプリケーション、バックアップ、 Snapshot 、クラスタなど、さまざまな Astra Control リソースに対する管理操作をサポートしています。

Astra SDK はパッケージの一部であり、単一の「 astrasdk.py 」ファイルで提供されています。このファイルを環境にインポートし、クラスを直接使用できます。

> NetApp Astra Control Python SDK * （または SDK のみ）は、パッケージ全体の名前です。**Astra SDK** は、単一ファイル「 astrasdk.py 」の中核的な Python クラスを指します。

Toolkit スクリプト

Astra SDK ファイルに加えて 'toolkit.py スクリプトも利用できますこのスクリプトは、 Python 関数として内部的に定義されている個別の管理アクションにアクセスできるようにすることで、抽象化のレベルを高めて動作します。このスクリプトは、 Astra SDK をインポートし、必要に応じてクラスに呼び出します。

にアクセスする方法

SDK には、次の方法でアクセスできます。

Python パッケージ

SDK は、から入手できます "Python パッケージインデックス" 名前の下にある * NetApp-Astra - ツールキット * 。パッケージにはバージョン番号が割り当てられており、必要に応じて更新が続行されます。パッケージを環境にインストールするには、 * PIP * パッケージ管理ユーティリティを使用する必要があります。

を参照してください "PyPi ： NetApp Astra Control Python SDK" を参照してください。

GitHub ソースコード

SDK ソースコードは GitHub からも入手できます。リポジトリには次のものが含まれます。

- `astrasdk.py (Python クラスを含む Astra SDK)
- `toolkit.py (より高度な関数ベースのスクリプト)
- インストール要件と手順の詳細
- インストールスクリプト
- その他のドキュメント

のクローンを作成できます "GitHub ： NetApp / NetApp-Astra - ツールキット" ローカル環境へのリポジトリ。

インストールと基本的な要件

パッケージのインストールおよび使用準備の一環として、いくつかのオプションと要件を考慮する必要があります。

インストールオプションの概要

次のいずれかの方法で SDK をインストールできます。

- Pip を使用して、 PyPI のパッケージを Python 環境にインストールします
- Git Hub リポジトリをクローニングし、次のいずれかを実行します。
  - パッケージを Docker コンテナとして導入（必要なものをすべて含む）
  - Python クライアントコードにアクセスできるように、 2 つのコア Python ファイルをコピーします

詳細については、 PyPi および GitHub のページを参照してください。

Astra Control 環境の要件

Astra SDK の Python クラスを直接使用する場合も、「 toolkit.py 」スクリプトの関数を使用する場合も、最終的には Astra Control 配置で REST API にアクセスすることになります。そのため、 API トークンを持つ Astra アカウントが必要になります。を参照してください "作業を開始する前に" 詳細については、このドキュメントの「 * はじめに * 」セクションの他のページを参照してください。

NetApp Astra Control Python SDK の要件

SDK には、ローカル Python 環境に関連するいくつかの前提条件があります。たとえば、 Python 3.5 以降を使用する必要があります。また、必要な Python パッケージもいくつかあります。詳細については、 GitHub リポジトリページまたは PyPI パッケージページを参照してください。

役立つリソースの概要

開始するために必要なリソースをいくつかご紹介します。

- "PyPi ： NetApp Astra Control Python SDK"
- "GitHub ： NetApp / NetApp-Astra - ツールキット"

ネイティブ **Python**

作業を開始する前に

Python は、特にデータセンターの自動化に広く使用されている開発言語です。Python のネイティブ機能をいくつかの共通パッケージとともに使用する前に、環境と必要な入力ファイルを準備する必要があります。

> ネットアップは、 Python を使用して Astra Control REST API に直接アクセスするだけでなく、 API を抽象化して複雑さを排除するツールキットパッケージも提供しています。を参照してください "NetApp Astra Control Python SDK" を参照してください。

環境を準備

以下に、 Python スクリプトを実行するための基本的な設定要件を示します。

Python 3.

最新バージョンの Python 3 がインストールされている必要があります。

追加ライブラリ

*Requests * および *ullib3 * ライブラリがインストールされている必要があります。環境に応じて、 pip などの Python 管理ツールを使用できます。

ネットワークアクセス

スクリプトを実行するワークステーションは、ネットワークにアクセスし、 Astra Control にアクセスできる必要があります。Astra Control Service を使用する場合は、インターネットに接続し、 https://astra.netapp.io` でサービスに接続できる必要があります。

ID 情報

アカウント ID と API トークンを持つ有効な Astra アカウントが必要です。を参照してください "API トークンを取得します" を参照してください。

JSON 入力ファイルを作成します

Python スクリプトは、 JSON 入力ファイルに含まれている設定情報に依存します。サンプルファイルは以下のとおりです。

> 環境に応じて、サンプルを更新する必要があります。

ID 情報

次のファイルに、 API トークンと Astra アカウントが含まれています。このファイルは '-i (または --identity') CLI パラメータを使用して Python スクリプトに渡す必要があります

```
{
  "api_token": "kH4CA_uVIa8q9UuPzhJaAHaGlaR7-no901DkkrVjIXk=",
  "account_id": "5131dfdf-03a4-5218-ad4b-fe84442b9786"
}
```

管理対象アプリケーションをリストします

次のスクリプトを使用して、 Astra アカウントの管理対象アプリケーションを一覧表示できます。

を参照してください "作業を開始する前に" 必要な JSON 入力ファイルの例を次に示します。

```
#!/usr/bin/env python3
##-----------------------------------------------------------------------
------
#
# Usage: python3 list_man_apps.py -i identity_file.json
#
# (C) Copyright 2021 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
#
##-----------------------------------------------------------------------
------

import argparse
import json
import requests
import urllib3
import sys

# Global variables
api_token = ""
account_id = ""

def get_managed_apps():
    ''' Get and print the list of managed apps '''

    # Global variables
    global api_token
    global account_id
```

```
    # Create an HTTP session
    sess1 = requests.Session()

    # Suppress SSL unsigned certificate warning
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

    # Create URL
    url1 = "https://astra.netapp.io/accounts/" + account_id +
"/k8s/v1/managedApps"

    # Headers and response output
    req_headers  = {}
    resp_headers = {}
    resp_data    = {}

    # Prepare the request headers
    req_headers.clear
    req_headers['Authorization'] = "Bearer " + api_token
    req_headers['Content-Type'] = "application/astra-managedApp+json"
    req_headers['Accept'] = "application/astra-managedApp+json"

    # Make the REST call
    try:
        resp1 = sess1.request('get', url1, headers=req_headers,
allow_redirects=True, verify=False)

    except requests.exceptions.ConnectionError:
        print("Connection failed")
        sys.exit(1)

    # Retrieve the output
    http_code = resp1.status_code
    resp_headers = resp1.headers

    # Print the list of managed apps
    if resp1.ok:
        resp_data = json.loads(resp1.text)
        items = resp_data['items']
        for i in items:
            print(" ")
            print("Name: " + i['name'])
            print("ID: " + i['id'])
            print("State: " + i['state'])
    else:
        print("Failed with HTTP status code: " + str(http_code))
```

```
    print(" ")

    # Close the session
    sess1.close()

    return

def read_id_file(idf):
    ''' Read the identity file and save values '''

    # Global variables
    global api_token
    global account_id

    with open(idf) as f:
        data = json.load(f)

    api_token = data['api_token']
    account_id = data['account_id']

    return

def main(args):
    ''' Main top level function '''

    # Global variables
    global api_token
    global account_id

    # Retrieve name of JSON input file
    identity_file = args.id_file

    # Get token and account
    read_id_file(identity_file)

    # Issue REST call
    get_managed_apps()

    return

def parseArgs():
        ''' Parse the CLI input parameters '''

        parser = argparse.ArgumentParser(description='Astra REST API -
List the managed apps',
                                add_help = True)
        parser.add_argument("-i", "--identity", action="store", dest
```

```
="id_file", default=None,
                            help='(Req) Name of the identity input file',
required=True)

        return parser.parse_args()

if __name__ == '__main__':
    ''' Begin here '''

    # Parse input parameters
    args = parseArgs()

    # Call main function
    main(args)
```

API リファレンス

HTTP メソッド、入力パラメータ、応答など、 Astra Control REST API のすべての呼び出しの詳細にアクセスできます。このリファレンスは、 REST API を使用して自動化アプリケーションを開発する場合に役立ちます。

> REST API のリファレンスドキュメントは、現在のところ Astra Control に付属しており、オンラインで入手できます。

作業を開始する前に

Astra Control Center または Astra Control Service のアカウントが必要です。

手順

1. アカウントのクレデンシャルを使用して Astra にサインインします。

   Astra Control Service の次のサイトにアクセスします。 "https://astra.netapp.io"

2. ページの右上にある図のアイコンをクリックし、 * API access * を選択します。
3. ページの上部で、 * API ドキュメント * の下に表示される URL をクリックします。
4. プロンプトが表示されたら、アカウントのクレデンシャルを再度入力し

その他のリソース

ここでは、 NetApp クラウドサービスとサポート、および REST とクラウドの一般的な概念について、ヘルプや詳細情報を参照するためのリソースを提供しています。

アストラ

- "Astra Control Center 22.04 のドキュメント"

  お客様の施設に導入されている Astra Control Center ソフトウェアの最新リリースのドキュメント。

- "Astra Control Service のマニュアル"

  パブリッククラウドで利用可能な最新リリースの Astra Control Service ソフトウェアのドキュメント。

- "Astra Trident のドキュメント"

  ネットアップが保守しているオープンソースのストレージオーケストレーションツールである Astra Trident ソフトウェアの最新リリースのドキュメント。

- "Astra ファミリーのドキュメント"

  オンプレミス環境とパブリッククラウド環境の両方で、すべての Astra ドキュメントに一元的にアクセスできます。

ネットアップのクラウドリソース

- "ネットアップのクラウドソリューション"

  ネットアップのクラウドソリューションの中心的なサイト。

- "NetApp Cloud Central コンソール"

  NetApp Cloud Central サービスコンソールにサインインします。

- "ネットアップサポート"

  トラブルシューティングツール、ドキュメント、およびテクニカルサポートにアクセスできます。

REST とクラウドの概念

- 博士 "失策" 著者 : Roy Fielding オリジナル版を読む

  本ドキュメントでは、 REST アプリケーション開発モデルを導入および確立しました。

- "Auth0"

  これは、 Astra サービスが Web アクセスに使用する認証および認可プラットフォームサービスです。

- "RFC エディタ"

  独自に番号付けされた RFC 文書の集合として維持されている Web およびインターネット標準の信頼できるソース。

旧バージョンの Astra Control Automation のドキュメント

以前の Astra Control リリースの自動化に関するドキュメントには、次のリンクからアクセスできます。

- "Astra Control Automation 21.12 ドキュメント"
- "Astra Control Automation 21.08 ドキュメント"

法的通知

著作権に関する声明、商標、特許などにアクセスできます。

著作権

"https://www.netapp.com/company/legal/copyright/"

商標

NetApp 、 NetApp のロゴ、および NetApp の商標ページに記載されているマークは、 NetApp, Inc. の商標です。その他の会社名および製品名は、それぞれの所有者の商標である場合があります。

"https://www.netapp.com/company/legal/trademarks/"

特許

ネットアップが所有する特許の最新リストは、次のサイトで入手できます。

https://www.netapp.com/pdf.html?item=/media/11887-patentspage.pdf

プライバシーポリシー

"https://www.netapp.com/company/legal/privacy-policy/"

Astra Control API ライセンス

https://docs.netapp.com/us-en/astra-automation/media/astra-api-license.pdf