Astra Control Automation 22.04文档

Astra Automation 22.04

NetApp
December 04, 2023

目录

Astra Control Automation 22.04文档

发行说明

关于此版本

本站点的文档介绍了随 2022 年 4 月（22.04）版的 Astra Control REST API 以及相关自动化技术。特别是，此版本的 REST API 随相应的 Astra 控制中心和 Astra 控制服务 22.04 版本一起提供。

有关此版本以及先前版本的详细信息，请参见以下页面和站点：

- "Astra Control REST API 的新增功能"
- "REST 资源和端点"
- "Astra Control Center 22.04 文档"
- "Astra Control Service 文档"
- "早期版本的 Astra Automation 文档"

Astra Control REST API 的新增功能

NetApp 会定期更新 Astra Control REST API，为您提供新功能，增强功能和错误修复。

2022 年 4 月 26 日（22.04）

此版本包括对 REST API 的扩展和更新，以及增强的安全性和管理功能。

新增和增强的 Astra 资源

添加了两种新的资源类型：* 软件包 * 和 * 升级 * 。此外，已升级多个现有资源的版本。

具有命名空间粒度的增强型 RBAC

将角色绑定到关联用户时，您可以限制用户有权访问的命名空间。请参见 * 角色绑定 API" 参考和 "RBAC 安全性" 有关详细信息 …

删除存储分段

您可以在不再需要某个存储分段或此存储分段无法正常运行时将其删除。

支持 Cloud Volumes ONTAP

现在支持将 Cloud Volumes ONTAP 用作存储后端。

其他产品增强功能

这两种 Astra Control 产品实施还有几项额外的增强功能，包括：

- Astra 控制中心的通用传入

- AKS 中的专用集群
- 支持 Kubernetes 1.22
- 支持 VMware Tanzu 产品组合

请参见 Astra 控制中心和 Astra 控制服务文档站点上的 * 新增功能 * 页面。

相关信息

- "Astra 控制中心：新增功能"
- "Astra Control Service ： 新增功能"

2021 年 12 月 14 日（21.12）

此版本扩展了 REST API ，并对文档结构进行了更改，以便在未来的版本更新中更好地支持 Astra Control 的发展。

每个版本的 **Astra Control** 都有单独的 **Astra Automation** 文档

每个版本的 Astra Control 都包含一个独特的 REST API ，该 API 已根据特定版本的功能进行了增强和定制。每个版本的 Astra Control REST API 的文档以及相关的 GitHub 内容存储库现在均可从其自己的专用网站上获得。主文档站点 "Astra Control Automation" 始终包含最新版本的文档。请参见 "早期版本的 Astra Control Automation 文档" 有关先前版本的信息。

扩展 **REST** 资源类型

REST 资源类型的数量不断增加，重点是执行挂钩和存储后端。新资源包括：帐户，执行钩，钩源，执行钩覆盖，集群节点， 受管存储后端，命名空间，存储设备和存储节点。请参见 "Resources" 有关详细信息 …

NetApp Astra Control Python SDK

NetApp Astra Control Python SDK 是一个开源软件包，可以更轻松地为您的 Astra Control 环境开发自动化代码。其核心是 Astra SDK ，其中包含一组类，用于抽象化 REST API 调用的复杂性。此外，还提供了一个工具包脚本，用于通过包装和抽象化 Python 类来执行特定管理任务。请参见 "NetApp Astra Control Python SDK" 有关详细信息 …

2021 年 8 月 5 日（21.08）

此版本引入了新的 Astra 部署模式，并对 REST API 进行了重大扩展。

Astra 控制中心部署模式

除了作为公有云服务提供的现有 Astra 控制服务之外，此版本还包括 Astra 控制中心内部部署模式。您可以在站点上安装 Astra 控制中心来管理本地 Kubernetes 环境。两种 Astra Control 部署模式共享同一个 REST API ，但文档中会根据需要指出一些细微的差异。

扩展 **REST** 资源类型

通过 Astra Control REST API 访问的资源数量已大幅增加，许多新资源为内部 Astra Control Center 产品奠定了基础。新资源包括： ASUP ，授权，功能，许可证，设置， 订阅，存储分段，云，集群，受管集群， 存储后端和存储类。请参见 "Resources" 有关详细信息 …

支持 **Astra** 部署的其他端点

除了扩展的 REST 资源之外，还有其他几个新的 API 端点可用于支持 Astra Control 部署。

支持 **OpenAPI**

通过 OpenAPI 端点可以访问当前的 OpenAPI JSON 文档和其他相关资源。

支持 **OpenMetrics**

通过 OpenMetrics 资源，您可以通过 OpenMetrics 端点访问帐户指标。

2021 年 4 月 15 日（21.04）

此版本包含以下新增功能和增强功能：

引入 **REST API**

Astra Control REST API 可与 Astra Control Service 产品配合使用。它是基于 REST 技术和当前最佳实践创建的。API 为 Astra 部署的自动化奠定了基础，并具有以下功能和优势。

Resources

有 14 种可用的 REST 资源类型。

API 令牌访问

REST API 的访问通过 API 访问令牌提供，您可以在 Astra Web 用户界面上生成此令牌。通过 API 令牌，可以安全地访问 API 。

支持收集

有一组丰富的查询参数，可用于访问资源集合。支持的部分操作包括筛选，排序和分页。

已知问题

您应查看当前版本中与 Astra Control REST API 相关的所有已知问题。已知问题可确定可能会阻止您成功使用本产品的问题。

> Astra Control REST API 22.04 版本没有新的已知问题。以下所述问题已在先前版本中发现，但仍适用于当前版本。

未发现后端存储节点中的所有存储设备

在发出REST API调用以检索存储节点中定义的存储设备时、并非所有设备都会返回。

Astra Control REST API 简介

Astra 控制中心和 Astra 控制服务提供了一个通用的 REST API ，您可以通过编程语言或 Curl 等实用程序直接访问该 API 。下面介绍了 API 的主要亮点和优势。

> 要访问 REST API ，您需要先登录到 Astra Web 用户界面并生成 API 令牌。您必须在每个 API 请求中包含令牌。

基于 **REST** 技术构建

Astra Control API 是使用 REST 技术和当前最佳实践创建的。核心技术包括 HTTP ， JSON 和 RBAC 。

支持两种 **Astra Control** 部署模式

Astra 控制服务在公有云环境中使用，而 Astra 控制中心则用于您的内部部署。有一个 REST API 支持这两种部署模式。

清晰地映射 **REST** 端点资源和对象模型

用于访问资源的外部 REST 端点映射到由 Astra 服务在内部维护的一致对象模型。对象模型采用实体关系（ ential-Relationship ， ER ）建模设计，有助于明确定义 API 操作和响应。

丰富的查询参数集

REST API 提供了一组丰富的查询参数，您可以使用这些参数来访问资源集合。支持的部分操作包括筛选，排序和分页。

与 **Astra Control Web UI** 对齐

Astra Web 用户界面的设计与 REST API 一致，因此两个访问路径和用户体验之间保持一致。

强大的调试和问题确定数据

Astra Control REST API 可提供强大的调试和问题确定功能，包括系统事件和用户通知。

工作流进程

我们提供了一组工作流来帮助您开发自动化代码。这些工作流分为两大类：基础架构和管理。

高级自动化技术的基础

除了直接访问 REST API 之外，您还可以使用基于 REST API 的其他自动化技术。

Astra 系列文档的一部分

Astra Control Automation 文档是较大的 Astra 系列文档的一部分。请参见 "Astra 文档" 有关详细信息 …

入门

开始之前

您可以通过查看以下步骤快速准备开始使用 Astra Control REST API 。

拥有 **Astra** 帐户凭据

您需要使用 Astra 凭据登录到 Astra Web 用户界面并生成 API 令牌。使用 Astra 控制中心，您可以在本地管理这些凭据。使用 Astra 控制服务时，帐户凭据可通过 * 身份验证 0* 服务进行访问。

熟悉 **Kubernetes** 的基本概念

您应熟悉 Kubernetes 的几个基本概念。请参见 "基本 Kubernetes 概念" 有关详细信息 …

查看 **REST** 概念和实施

请务必查看 "核心 REST 实施" 有关 REST 概念的信息以及有关如何设计 Astra Control REST API 的详细信息。

获取更多信息

您应了解中建议的追加信息资源 "其他资源"。

获取 **API** 令牌

要使用 Astra Control REST API ，您需要获取 Astra API 令牌。

简介

API 令牌用于标识 Astra 的调用方，并且必须包含在每个 REST API 调用中。

- 您可以使用 Astra Web 用户界面生成 API 令牌。
- 令牌附带的用户身份由创建令牌的用户确定。
- 令牌必须包含在 `Authorization` HTTP Request 标头中。
- 令牌创建后永不过期。
- 您可以在 Astra Web 用户界面上撤消令牌。

相关信息

- "撤消 API 令牌"

创建 **Astra API** 令牌

以下步骤介绍如何创建 Astra API 令牌。

开始之前

您需要 Astra 帐户的凭据。

关于此任务

此任务会在 Astra Web 界面上生成 API 令牌。此外，您还应检索帐户 ID ，在进行 API 调用时也需要使用此 ID 。

步骤

1. 使用您的帐户凭据登录到 Astra 。

   访问以下站点以获取 Astra 控制服务： "https://astra.netapp.io"

2. 单击页面右上角的图图标并选择 * API access* 。
3. 单击页面上的 * 生成 API 令牌 * ，然后在弹出窗口中单击 * 生成 API 令牌 * 。
4. 单击图标将令牌字符串复制到剪贴板并将其保存在编辑器中。
5. 复制并保存同一页面上提供的帐户 ID 。

完成后

通过 Curl 或编程语言访问 Astra Control REST API 时，必须在 HTTP `Authorization` Request 标头中包含 API 承载令牌。

大家好

您可以在工作站的命令行界面上通过问题描述执行一个简单的 Curl 命令来开始使用 Astra Control REST API 并确认其可用性。

开始之前

本地工作站必须提供 Curl 实用程序。您还必须具有 API 令牌和关联的帐户标识符。请参见 "获取 API 令牌" 有关详细信息 …

curl 示例

以下 Curl 命令将检索 Astra 用户列表。按指示提供相应的 <account_ID> 和 <api_token> 。

```
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 输出示例

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

准备使用这些工作流

在将 Astra 工作流用于实时部署之前，您应熟悉这些工作流的组织和格式。

简介

_Workflow _ 是完成特定管理任务或目标所需的一个或多个步骤的序列。Astra Control 工作流中的每个步骤均为以下步骤之一：

- REST API 调用（包含 curl 和 JSON 示例等详细信息）
- 调用另一个 Astra 工作流
- 其他相关任务（例如做出必要的设计决策）

这些工作流包括完成每个任务所需的核心步骤和参数。它们为自定义自动化环境提供了一个起点。

通用输入参数

以下所述的输入参数对于用于说明 REST API 调用的所有 curl 示例通用。

> 由于这些输入参数是通用的要求，因此在各个工作流中不会对其进行进一步说明。如果在特定的 curl 示例中使用了其他输入参数，则会在 * 其他输入参数 * 一节中进行介绍。

路径参数

每次 REST API 调用使用的端点路径都包括以下参数。另请参见 "URL 格式" 有关详细信息 …

帐户 **ID**

这是用于标识运行 API 操作的 Astra 帐户的 UUIDv4 值。请参见 "获取 API 令牌" 有关查找帐户 ID 的详细信息，请参见。

请求标题

根据 REST API 调用，您可能需要包含多个请求标头。

Authorization

工作流中的所有 API 调用都需要 API 令牌来标识用户。您必须在 `Authorization` Request 标题中包含令牌。请参见 "获取 API 令牌" 有关生成 API 令牌的详细信息。

内容类型

对于请求正文中包含 JSON 的 HTTP POST 和 PUT 请求，您应根据 Astra 资源声明介质类型。例如，在为受管应用程序创建快照时，可以包括标题 `Content-Type ： application/Astra-appsnap+json` 。

接受

您可以根据 Astra 资源声明响应中预期内容的特定介质类型。例如，在列出受管应用程序的备份时，您可以包括标题 `accept ： application/Astra-appBackup+json` 。但是，为了简单起见，工作流中的 curl 样本接受所有介质类型。

表示令牌和标识符

在 curl 示例中使用的 API 令牌和其他 ID 值是不透明的，没有明显的含义。因此，为了提高示例的可读性，不会使用实际令牌和 ID 值。而是使用较小的保留关键字，它具有以下几个优势：

- curl 和 JSON 示例更清晰，更易于理解。
- 由于所有关键字都使用相同的格式以及括号和大写字母，因此您可以快速确定要插入或提取的位置和内容。
- 不会丢失任何值，因为无法复制原始参数并将其用于实际部署。

以下是在 curl 示例中使用的一些常见保留关键字。此列表并非详尽无遗，我们会根据需要使用其他关键字。根据具体情况，其含义应该是显而易见的。

关键字	**Type**	**Description**
<account_ID>	路径	用于标识运行 API 操作的帐户的 UUIDv4 值。
<api_token>	标题	标识和授权调用方的令牌。
<managed_app_ID>	路径	用于标识 API 调用的受管应用程序的 UUIDv4 值。

工作流类别

根据您的部署模式，有两大类 Astra 工作流可用。如果您使用的是 Astra 控制中心，则应先从基础架构工作流开始，然后再继续执行管理工作流。使用 Astra Control Service 时，通常可以直接转到管理工作流。

> 工作流中的 curl 示例使用 Astra 控制服务的 URL 。在根据您的环境使用内部 Astra 控制中心时，您需要更改 URL 。

基础架构工作流

这些工作流用于处理 Astra 基础架构，包括凭据，存储分段和存储后端。Astra 控制中心需要使用这些控制器，但在大多数情况下，也可以与 Astra 控制服务一起使用。这些工作流侧重于建立和维护 Astra 受管集群所需的任务。

管理工作流

您可以在拥有受管集群后使用这些工作流。这些工作流侧重于应用程序保护和支持操作，例如备份，还原和克隆受管应用程序。

基本 Kubernetes 概念

使用 Astra REST API 时，有几个相关的 Kubernetes 概念。

对象

Kubernetes 环境中维护的对象是表示集群配置的永久性实体。这些对象共同描述了系统的状态，包括集群工作负载。

命名空间

命名空间提供了一种隔离单个集群中的资源的技术。在划分工作类型，用户类型和资源类型时，此组织结构非常有用。命名空间范围 _ 的对象在命名空间中必须是唯一的，而具有 _cluster 范围 _ 的对象必须在整个集群中是唯一的。

标签

标签可以与 Kubernetes 对象关联。它们使用键值对描述属性，并可在集群上强制实施任意组织，这对组织可能有用，但不在核心 Kubernetes 操作范围内。

核心 REST 实施

REST Web 服务

表述性状态传输（ Representational State Transfer ， REST ）是一种用于创建分布式 Web 应用程序的模式。在设计 Web 服务 API 时，它会建立一组用于公开基于服务器的资源并管理其状态的主流技术和最佳实践。虽然 REST 为应用程序开发提供了一致的基础，但每个 API 的详细信息可能因特定设计选项而异。在将 Astra Control REST API 用于实时部署之前，您应了解其特征。

资源和状态表示

资源是基于 Web 的系统的基本组件。创建 REST Web 服务应用程序时，早期设计任务包括：

- 识别系统或基于服务器的资源

  每个系统都使用和维护资源。资源可以是文件，业务事务，流程或管理实体。在设计基于 REST Web 服务的应用程序时，首先要完成的任务之一是识别资源。

- 资源状态和关联状态操作的定义

  资源始终处于数量有限的状态之一。必须明确定义状态以及用于影响状态更改的关联操作。

URI 端点

必须使用定义明确的寻址方案定义和提供每个 REST 资源。资源所在的端点和标识的端点使用统一资源标识符（ Uniform Resource Identifier ， URI ）。URI 提供了一个通用框架，用于为网络中的每个资源创建唯一名称。统一资源定位器（ Uniform Resource Locator ， URL ）是一种用于 Web 服务的 URI 类型，用于标识和访问资源。资源通常以类似于文件目录的分层结构公开。

HTTP 消息

超文本传输协议（ HTTP ）是 Web 服务客户端和服务器用来交换有关资源的请求和响应消息的协议。在设计 Web 服务应用程序时， HTTP 方法会映射到资源以及相应的状态管理操作。HTTP 为无状态。因此，要将一组相关请求和响应关联为一个事务的一部分，必须将追加信息包含在随请求和响应数据流一起提供的 HTTP 标头中。

JSON 格式化

虽然信息可以通过多种方式在 Web 服务客户端和服务器之间进行结构化和传输，但最受欢迎的选项是 JavaScript 对象表示法（ JSON ）。JSON 是一种行业标准，用于以纯文本形式表示简单数据结构，并用于传输描述资源的状态信息。Astra Control REST API 使用 JSON 格式化每个 HTTP 请求和响应正文中包含的数据。

资源和集合

通过 Astra Control REST API ，可以访问资源实例和资源实例集合。

> 从概念上讲， REST * 资源 * 类似于使用面向对象的编程（ OOP ）语言和系统定义的 * 对象 * 。有时，这些术语可以互换使用。但一般来说，在外部 REST API 环境中使用时，首选使用 "resource" ，而在服务器上存储的相应有状态实例数据中使用 "object" 。

Astra 资源的属性

Astra Control REST API 符合 RESTful 设计原则。每个 Astra 资源实例都是根据定义明确的资源类型创建的。一组相同类型的资源实例称为 * 集合 * 。API 调用会对单个资源或资源集合执行操作。

资源类型

Astra Control REST API 附带的资源类型具有以下特征：

- 每个资源类型均使用模式定义（通常在 JSON 中）
- 每个资源架构都包括资源类型和版本
- 资源类型在全局范围内是唯一的

资源实例

通过 Astra Control REST API 提供的资源实例具有以下特征：

- 资源实例是根据单个资源类型创建的
- 资源类型使用介质类型值来指示
- 实例由由 Astra 服务维护的有状态数据组成
- 每个实例均可通过一个唯一且长期存在的 URL 进行访问
- 如果某个资源实例可以具有多个表示形式，则可以使用不同的介质类型来请求所需的表示形式

资源收集

通过 Astra Control REST API 提供的资源收集具有以下特征：

- 一种资源类型的一组资源实例称为集合
- 资源集合具有一个唯一且长期存在的 URL

实例标识符

创建每个资源实例时，系统都会为其分配一个标识符。此标识符是一个 128 位 UUIDv4 值。分配的 UUIDv4 值是全局唯一且不可变的。发出创建新实例的 API 调用后， HTTP 响应的 `Location` 标头中会将具有关联 ID 的 URL 返回给调用方。在引用资源实例时，您可以提取此标识符并在后续调用中使用它。

> 资源标识符是用于收集的主密钥。

Astra 资源的通用结构

每个 Astra Control 资源都使用一个通用结构进行定义。

通用数据

每个 Astra 资源都包含下表所示的键值。

密钥	Description
type	一种全局唯一资源类型，称为 * 资源类型 * 。
version	一种称为 * 资源版本 * 的版本标识符。
id	一种全局唯一标识符，称为 * 资源标识符 * 。
元数据	一个 JSON 对象，包含各种信息，包括用户和系统标签。

元数据对象

每个 Astra 资源附带的元数据 JSON 对象包含下表所示的键值。

密钥	Description
labels	与资源关联的客户端指定标签的 JSON 数组。
creationTimestamp	JSON 字符串，其中包含指示资源创建时间的时间戳。
modificationTimestamp	JSON 字符串，其中包含 ISO-8601 格式的时间戳，用于指示资源上次更改的时间。
已创建	JSON 字符串，其中包含创建资源的用户 ID 的 UIDv4 标识符。如果资源是由内部系统组件创建的，并且没有与创建实体关联的 UUID ，则使用 * 空 * UUID 。

资源状态

选定资源 A `state` 值，用于编排生命周期过渡和控制访问。

HTTP 详细信息

Astra Control REST API 使用 HTTP 以及相关参数对资源和集合执行操作。下面提供了 HTTP 实施的详细信息。

API 事务和 CRUD 模型

Astra Control REST API 可实施一个事务模式，其中包含定义明确的操作和状态过渡。

请求和响应 API 事务

每次 REST API 调用都是作为对 Astra 服务的 HTTP 请求执行的。每个请求都会向客户端生成关联的响应。此请求响应对可视为 API 事务。

支持 CRUD 操作模式

通过 Astra Control REST API 提供的每个资源实例和集合均可根据 * CRU* 模型进行访问。有四个操作，每个操作都映射到一个 HTTP 方法。这些操作包括：

- 创建
- 读取
- 更新
- 删除

对于某些 Astra 资源，仅支持其中一部分操作。您应查看 "API 参考" 有关特定 API 调用的详细信息。

HTTP 方法

下表显示了 API 支持的 HTTP 方法或动词。

方法	CRUD	Description
获取	读取	检索资源实例或集合的对象属性。在与集合结合使用时，此操作被视为 * 列表 * 操作。
发布	创建	根据输入参数创建新的资源实例。长期 URL 会在 `Location` 响应标头中返回。
PUT	更新	使用提供的 JSON 请求正文更新整个资源实例。系统会保留用户不可修改的密钥值。
删除	删除	删除现有资源实例。

请求和响应标头

下表汇总了与 Astra Control REST API 一起使用的 HTTP 标头。

请参见 "RFC 7232" 和 "RFC 7233" 有关详细信息 …

标题	Type	使用说明
接受	请求	如果值为 "/" 或未提供，则在内容类型响应标题中返回 `application/json` 。如果此值设置为 Astra 资源的介质类型，则内容类型标题中将返回相同的介质类型。
Authorization	请求	包含用户 API 密钥的承载令牌。
内容类型	响应	根据 `accept` request 标头返回。
ETAG	响应	随 RFC 7232 中定义的成功附带。该值是整个 JSON 资源的 MD5 值的十六进制表示形式。
如果匹配	请求	一个前提条件请求标头，如第 3.1 节 RFC 7232 中所述实施，并支持 * PUT * 请求。
if-Modified-since	请求	一个前提条件请求标头，如第 3.4 节 RFC 7232 中所述实施，并支持 * PUT * 请求。
如果未修改，则从	请求	一个前提条件请求标头，如第 3.4 节 RFC 7232 中所述实施，并支持 * PUT * 请求。
位置	响应	包含新创建资源的完整 URL 。

查询参数

以下查询参数可用于资源收集。请参见 "使用收集" 有关详细信息 …

查询参数	Description
包括	包含读取收集时应返回的字段。

查询参数	Description
筛选器	指示读取收集时要返回的资源必须匹配的字段。
订单	确定读取收集时返回的资源的排序顺序。
limit	限制读取集合时返回的最大资源数。
跳过	设置读取集合时要传递和跳过的资源数量。
count	指示是否应在元数据对象中返回资源总数。

HTTP status codes

下面介绍了 Astra Control REST API 使用的 HTTP 状态代码。

> Astra Control REST API 还使用 * HTTP APIs* 标准的问题详细信息。请参见 "诊断和支持" 有关详细信息 …

代码	含义	Description
200	确定	表示未创建新资源实例的调用成功。
201	已创建	已成功创建对象，并且位置响应标头包含该对象的唯一标识符。
204	No Content	尽管未返回任何内容，但请求成功。
400	请求错误	此请求输入无法识别或不适当。
401	未授权	用户未获得授权，必须进行身份验证。
403	已禁止	由于授权错误，访问被拒绝。
404	未找到	请求中引用的资源不存在。
409	冲突	尝试创建对象失败，因为此对象已存在。
500	内部错误	服务器发生一般内部错误。
503	服务不可用	由于某种原因，此服务尚未准备好处理此请求。

URL 格式

用于通过 REST API 访问资源实例或集合的 URL 的常规结构由多个值组成。此结构反映了底层对象模型和系统设计。

帐户作为 root

指向每个 REST 端点的资源路径的根是 Astra 帐户。因此， URL 中的所有路径均以 ` /account/ ｛account_id｝ ` 开头，其中 `account_id` 是帐户的唯一 UUIDv4 值。内部结构这反映了一种设计，其中所有资源访问都基于特定帐户。

端点资源类别

Astra 资源端点分为三类：

- 核心（` / 核心` ）

- 受管应用程序（` /K8s` ）
- 拓扑（` / 拓扑` ）

请参见 "Resources" 有关详细信息 …

类别版本

这三个资源类别中的每一个都有一个全局版本，用于控制所访问资源的版本。按照约定和定义，迁移到资源类别的新主要版本（例如，从 ` /v1` 到 ` /v2` ）将会导致 API 发生中断更改。

资源实例或集合

根据是否访问资源实例或集合，可以在路径中使用资源类型和标识符的组合。

示例

- 资源路径

  根据上述结构，端点的典型路径为：` /accouns/ ｛ account_id ｝ /core/v1/users` 。

- 完整的 URL

  相应端点的完整 URL 为： `https://astra.netapp.io/accounts/{account_id}/core/v1/users` 。

资源和端点

您可以使用通过 Astra Control REST API 提供的资源来自动执行 Astra 部署。每个资源都通过一个或多个端点进行访问。下面提供的信息介绍了可在自动化部署中使用的 REST 资源。

> 用于访问 Astra Control 资源的路径和完整 URL 的格式基于多个值。请参见 "URL 格式" 有关详细信息 …另请参见 "API 参考" 有关使用 Astra 资源和端点的更多详细信息。

Astra Control REST 资源摘要

Astra Control REST API 中提供的主要资源端点分为三类。除了需要说明的情况外，可以使用一整套 CRUD 操作（创建，读取，更新，删除）来访问每个资源。

- 版本 * 列表示首次引入资源时的 Astra 版本。对于新添加到当前版本的资源，此字段为粗体。

核心资源

核心资源端点可提供建立和维护 Astra 运行时环境所需的基础服务。

资源	版本。	**Description**
Account	21.12	通过帐户资源，您可以管理多租户 Astra Control 部署环境中的隔离租户。
ASUP	21.08.	ASUP 资源表示转发给 NetApp 支持部门的 AutoSupport 捆绑包。
凭据	21.04	凭据资源包含可用于 Astra 用户，集群，分段和存储后端的安全相关信息。
授权	21.08.	授权资源表示根据活动许可证和订阅可供帐户使用的功能和容量。
事件	21.04	事件资源表示系统中发生的所有事件，包括归类为通知的子集。
执行钩	21.12	执行钩资源表示自定义脚本，您可以在执行受管应用程序的快照之前或之后运行这些脚本。
功能	21.08.	这些功能资源表示选定的 Astra 功能，您可以查询这些功能来确定它们是否已在系统中启用。访问权限仅限于只读。
挂钩源	21.12	hook 源资源表示与执行 hook 一起使用的实际源代码。将源代码与执行控制分开具有多种优势，例如允许共享脚本。
许可证	21.08.	许可证资源表示可用于 Astra 帐户的许可证。
通知	21.04	通知资源表示具有通知目标的 Astra 事件。访问权限按用户提供。
软件包	* 。 22.04*	软件包资源用于注册和访问软件包定义。软件包由多个组件组成，包括文件，映像和其他项目。
角色绑定	21.04	角色绑定资源表示特定用户对和帐户之间的关系。除了这两者之间的链接之外，还会通过特定角色为每个指定一组权限。
正在设置 …	21.08.	设置资源表示一组密钥值对，用于描述特定 Astra 帐户的功能。
订阅。	21.08.	订阅资源表示 Astra 帐户的活动订阅。
令牌	21.04	令牌资源表示可通过编程方式访问 Astra Control REST API 的令牌。

资源	版本。	Description
未读通知	21.04	未读通知资源表示已分配给特定用户但尚未读取的通知。
升级	* 。 22.04*	通过升级资源，您可以访问软件组件并启动升级。
用户	21.04	用户资源表示 Astra 用户能够根据其定义的角色访问系统。

受管应用程序资源

通过受管应用程序资源端点，可以访问受管 Kubernetes 应用程序。

资源	版本。	Description
应用程序资产	21.04	应用程序资产资源表示管理 Astra 应用程序所需的内部状态信息集合。
应用程序备份	21.04	应用程序备份资源表示受管应用程序的备份。
应用程序快照	21.04	应用程序快照资源表示受管应用程序的快照。
执行挂机覆盖	21.12	使用执行挂钩覆盖资源，您可以根据需要为特定应用程序禁用预加载的 NetApp 默认执行挂钩。
受管应用程序	21.04	托管应用程序资源代表由 Astra 管理的 Kubernetes 应用程序。
计划	21.04	计划资源表示在数据保护策略中为受管应用程序计划的数据保护操作。

拓扑资源

通过拓扑资源端点可以访问非受管应用程序和存储资源。

资源	版本。	Description
应用程序	21.04	应用程序资源表示所有 Kubernetes 应用程序，包括那些不受 Astra 管理的应用程序。
存储分段	21.08.	存储分段资源表示用于存储由 Astra 管理的应用程序备份的 S3 云分段。
云	21.08.	云资源表示 Astra 客户端为管理集群和应用程序而可以连接到的云。
集群	21.08.	集群资源表示不受 Kubernetes 管理的 Kubernetes 集群。
集群节点	21.12	集群节点资源允许您访问 Kubernetes 集群中的各个节点，从而提供额外的解析。
受管集群	21.08.	受管集群资源表示当前由 Kubernetes 管理的 Kubernetes 集群。
托管存储后端	21.12	通过托管存储后端资源，您可以访问后端存储提供程序的抽象表示形式。这些存储后端可供受管集群和应用程序使用。
命名空间	21.12	命名空间资源用于访问 Kubernetes 集群中使用的命名空间。
存储后端	21.08.	存储后端资源表示可由 Astra 管理的集群和应用程序使用的存储服务提供商。
存储类	21.08.	存储类资源表示已发现并可供特定受管集群使用的不同存储类或类型。
Volume	21.04	卷资源表示与受管应用程序关联的 Kubernetes 存储卷。

最新版本中的新端点

当前的 22.04 Astra Control 版本增加了以下 REST 端点。此外，已升级多个现有资源的版本。

- /accouns/ { account_id } /core/v1/packages
- /accouns/ { account_id } /core/v1/packages/ { package_id }
- /accouns/ { account_id } /core/v1/upgrades
- /accouns/ { account_id } /core/v1/upgrades// { upgrade_id }
- /accouns/ { account_id } /topology/v1/appBackups
- /accouns/ { account_id } /topology/v1/appBackups/ { appBackup_id }
- /accouns/ { account_id } /topology/v1/cloud / { clune_id } /clusters/ { cluster_id } /clusterNode
- /accouns/ { account_id } /topology/v1/cloud / { clune_id } /cluster/ { cluster_id } /clusterNodes/ { clusterNode_id }
- /accouns/ { account_id } /topology/v1/managedClusters/ { managedCluster_id } /apps/ { app_id } /appAssets
- /accouns/ { account_id } /topology/v1/managedClusters/ { managedCluster_id } /apps/ { app_id } /appAssets/ { appasset_id }
- /accouns/ { account_id } /topology/v1/managedClusters/ { managedCluster_id } /clusterNode
- /accouns/ { account_id } /topology/v1/managedClusters/ { managedCluster_id } /clusterNodes/ { clusterNode_id }

其他资源和端点

您可以使用多种其他资源和端点来支持 Astra 部署。

这些资源和端点当前未包含在 Astra Control REST API 参考文档中。

OpenAPI

通过 OpenAPI 端点可以访问当前的 OpenAPI JSON 文档和其他相关资源。

OpenMetrics

通过 OpenMetrics 端点，您可以通过 OpenMetrics 资源访问帐户指标。Astra 控制中心部署模式支持此功能。

其他使用注意事项

RBAC 安全性

Astra REST API 支持基于角色的访问控制（Role-Based Access Control， RBAC）来限制对系统功能的访问。

Astra 角色

每个 Astra 用户都分配有一个角色，用于确定可执行的操作。这些角色按层次结构进行排列，如下表所述。

Role	Description
所有者	具有管理员角色的所有权限，并且还可以删除 Astra 帐户。
管理员	具有成员角色的所有权限，并且还可以邀请用户加入帐户。
成员	可以全面管理 Astra 应用程序和计算资源。
查看器	仅限查看资源。

具有命名空间粒度的增强型 RBAC

此功能是在 Astra REST API 22.04 版中引入的。

为特定用户建立角色绑定后，可以应用一个限制来限制用户有权访问的命名空间。可通过多种方法定义此限制，如下表所述。有关详细信息，请参见角色绑定 API 中的参数 `roleContrcons` 。

命名空间	Description
全部	用户可以通过通配符参数 "*" 访问所有命名空间。这是保持向后兼容性的默认值。
无	尽管约束列表为空，但仍会指定此限制列表。这表示用户无法访问任何命名空间。
命名空间列表	包含命名空间的 UUID ，这会将用户限制为单个命名空间。此外，还可以使用逗号分隔列表来访问多个命名空间。
Label	指定了一个标签，并允许访问所有匹配的命名空间。

使用收集

Astra Control REST API 提供了多种不同的方法来通过定义的查询参数访问资源收集。

选择值

您可以使用 `include` 参数为每个资源实例指定应返回的键值对。所有实例都会在响应正文中返回。

筛选

通过收集资源筛选， API 用户可以指定条件，以确定是否在响应正文中返回资源。`filter` 参数用于指示筛选条件。

排序

通过收集资源排序， API 用户可以在响应正文中指定资源的返回顺序。`ordersBy` 参数用于指示筛选条件。

分页

您可以通过使用 `limit` 参数限制为请求返回的资源实例数来强制分页。

计数

如果包括布尔参数 `count` set to `true` ，则元数据部分将提供给定响应返回数组中的资源数。

诊断和支持

Astra Control REST API 提供了多种支持功能，可用于诊断和调试。

API resources

API 资源提供了多种可提供诊断信息和支持的 Astra 功能。

Type	Description
事件	在 Astra 处理过程中记录的系统活动。
通知	被视为足够重要的一小部分事件，可提供给用户。
未读通知	用户尚未读取或检索的通知。

撤消 API 令牌

您可以在 Astra Web 界面上撤消不再需要的 API 令牌。

开始之前

您需要一个 Astra 帐户。您还应确定要撤消的令牌。

关于此任务

令牌撤消后，它将立即永久不可用。

步骤

1. 使用您的帐户凭据登录到 Astra 。

   访问以下站点以获取 Astra 控制服务： "https://astra.netapp.io"

2. 单击页面右上角的图图标并选择 * API access* 。
3. 选择要撤消的一个或多个令牌。
4. 在 * 操作 * 下拉框下，单击 * 撤消令牌 * 。

基础架构工作流

开始之前

您可以使用这些工作流创建和维护与 Astra 控制中心部署模式结合使用的基础架构。在大多数情况下，这些工作流也可与 Astra Control Service 结合使用。

NetApp 可以随时扩展和改进这些工作流，因此您应定期查看这些工作流。

一般准备

在使用任何 Astra 工作流之前，请务必查看 "准备使用这些工作流"。

工作流类别

基础架构工作流按不同类别进行组织，以便更容易找到所需的工作流。

类别	Description
身份和访问	通过这些工作流，您可以管理身份以及如何访问 Astra 。这些资源包括用户，凭据和令牌。
存储分段	您可以使用这些工作流创建和管理用于存储备份的 S3 存储分段。
存储	通过这些工作流，您可以添加和维护存储后端和卷。
集群	您可以添加受管 Kubernetes 集群，以便保护和支持这些集群所包含的应用程序。

身份和访问

列出用户

您可以列出为特定 Astra 帐户定义的用户。

1. 列出用户

执行以下 REST API 调用。

HTTP 方法	路径
获取	/account/ ｛accountID｝ /core/v1/users

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	**Type**	**Required**	**Description**
包括	查询	否	也可以选择要在响应中返回的值。

curl 示例：返回所有用户的所有数据

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

curl 示例：返回所有用户的名字，姓氏和 ID

```
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 输出示例

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

存储分段

列出分段

您可以列出为特定 Astra 帐户定义的 S3 存储分段。

1. 列出分段

执行以下 REST API 调用。

HTTP 方法	路径
获取	/account/ ｛accountID｝ /topology/v1/b桶

curl 示例：返回所有存储分段的所有数据

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

存储

列出存储后端

您可以列出可用的存储后端。

1. 列出分段

执行以下 REST API 调用。

HTTP 方法	路径
获取	/account/ ｛accountID｝ /topology/v1/storageBackend

curl 示例：返回所有存储后端的所有数据

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

JSON 输出示例

```
{
  "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"
    }
  ]
}
```


集群

列出受管集群

您可以列出当前由 Astra 管理的 Kubernetes 集群。

1. 列出集群

执行以下 REST API 调用。

HTTP 方法	路径
获取	/account/ ｛accountID｝ /topology/v1/managedClusters

curl 示例：返回所有集群的所有数据

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

管理工作流

开始之前

您可以在管理 Astra 受管集群中的应用程序时使用这些工作流。

NetApp 可以随时扩展和改进这些工作流，因此您应定期查看这些工作流。

一般准备

在使用任何 Astra 工作流之前，请务必查看 "准备使用这些工作流"。

工作流类别

管理工作流按不同类别进行组织，以便更容易找到所需的工作流。

类别	**Description**
应用程序控制	通过这些工作流，您可以控制受管应用程序和非受管应用程序。您可以列出应用程序以及创建和删除受管应用程序。
应用程序保护	您可以使用这些工作流通过快照和备份保护受管应用程序。
克隆和还原应用程序	这些工作流介绍了如何克隆和还原受管应用程序。
支持	有多个工作流可用于调试和支持您的应用程序以及常规 Kubernetes 环境。

其他注意事项

使用管理工作流时，还需要注意一些其他注意事项。

克隆应用程序

克隆应用程序时，需要考虑一些事项。以下所述参数是 JSON 输入的一部分。

源集群标识符
: 值 `sClusterID` 始终表示安装原始应用程序的集群。

集群标识符
: 值 `clusterID` 用于标识要安装新应用程序的集群。

- 在同一集群中克隆时， `clusterID` 和 `sClusterID` 具有相同的值。
- 在集群间克隆时，这两个值不同，并且 `clusterID` 应为目标集群的 ID 。

命名空间
: `namespace` 值必须与原始源应用程序不同。此外，克隆的命名空间不存在， Astra 将创建它。

备份和快照

您可以选择使用 `backupID` 或 `snapshotID` 参数从现有备份或快照克隆应用程序。如果不提供备份或快照，则 Astra 将首先创建应用程序的备份，然后从备份中克隆。

还原应用程序

以下是还原应用程序时需要考虑的几个事项。

- 还原应用程序与克隆操作非常相似。
- 还原应用程序时，您必须提供备份或快照。

应用程序控制

列出非受管应用程序

您可以列出当前不受 Astra 管理的应用程序。您可以在选择要管理的应用程序时执行此操作。

> 默认情况下，这些工作流中使用的 REST 端点将返回所有 Astra 应用程序。您可以在 API 调用中使用 `filter` query 参数，仅请求返回非受管应用程序。或者，您也可以省略 filter 参数以返回所有应用程序，然后检查输出中的 `managedState` 字段以确定哪些应用程序处于 非受管 状态。

仅列出受管理状态等于非受管状态的应用程序

此工作流使用 `filter` query 参数仅返回非受管应用程序。

1. 列出非受管应用程序

执行以下 REST API 调用。

HTTP 方法	路径
获取	/account/ ｛accountID｝ /topology/v1/apps

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	**Type**	**Required**	**Description**
筛选器	查询	否	使用筛选器指定应返回哪些应用程序。
包括	查询	否	也可以选择要在响应中返回的值。

curl 示例：返回非受管应用程序的名称， **ID** 和管理状态

```
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 输出示例


```
{
  "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": {}
}
```

列出所有应用程序并选择非受管应用程序

此工作流将返回所有应用程序。您必须检查输出以确定哪些不受管。

1. 列出所有应用程序

执行以下 REST API 调用。

HTTP 方法	路径
获取	/account/ ｛accountID｝ /topology/v1/apps

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	**Type**	**Required**	**Description**
包括	查询	否	也可以选择要在响应中返回的值。

curl 示例：返回所有应用程序的所有数据

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

curl 示例：返回所有应用程序的名称， **ID** 和 **managedState**

```
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 输出示例

```
{
  "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. 选择非受管应用程序

查看 API 调用的输出，然后手动选择 `managedState` 等于 **非受管** 的应用程序。

列出受管应用程序

您可以列出当前由 Astra 管理的应用程序。您可以在查找特定应用程序的快照或备份时执行此操作。

1. 列出应用程序

执行以下 REST API 调用。

HTTP 方法	路径
获取	/account/ ｛accountID｝ /K8s/v1/managedApps

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	**Type**	**Required**	**Description**
包括	查询	否	也可以选择要在响应中返回的值。

curl 示例：返回所有应用程序的所有数据

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

curl 示例：返回所有应用程序的名称， **ID** 和状态

```
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 输出示例

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

获取托管应用程序

您可以检索描述单个受管应用程序的所有资源变量。

开始之前

您必须具有要检索的受管应用程序的 ID 。如果需要，您可以使用此工作流 "列出受管应用程序" 以查找应用程序。

1. 获取应用程序

执行以下 REST API 调用。

HTTP 方法	路径
获取	/accouns/ { account_id } /K8s/v1/managedApps/ { managedApp_id }

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	**Type**	**Required**	**Description**
受管应用程序 ID	路径	是的。	要检索的受管应用程序的 ID 值。

curl 示例：返回应用程序的所有数据

```
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>'
```

管理应用程序

您可以基于 Astra 已知的应用程序创建托管应用程序。管理应用程序时，您可以通过定期

备份和快照来对其进行保护。

开始之前

您必须具有要管理的已发现应用程序的 ID 。如果需要，您可以使用此工作流 "列出非受管应用程序" 以查找应用程序。

1. 管理应用程序

执行以下 REST API 调用。

HTTP 方法	路径
发布	/account/ ｛accountID｝ /K8s/v1/managedApps

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	**Type**	**Required**	**Description**
JSON	body	是的。	提供确定要管理的应用程序所需的参数。请参见以下示例。

JSON 输入示例

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

curl 示例：管理应用程序

```
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
```

取消管理应用程序

您可以删除不再需要的受管应用程序。删除受管应用程序也会删除关联的计划。

开始之前

您必须具有要取消管理的受管应用程序的 ID 。如果需要，您可以使用此工作流 "列出受管应用程序" 以查找应用程序。

删除应用程序时，不会自动删除其备份和快照。如果您不再需要备份和快照，应在删除应用程序之前将其删除。

1. 非受管应用程序

执行以下 REST API 调用。

HTTP 方法	路径
删除	/accouns/ { account_id } /K8s/v1/managedApps/ { managedApp_id }

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	**Type**	**Required**	**Description**
受管应用程序 ID	路径	是的。	标识要删除的受管应用程序。

curl 示例：删除受管应用程序

```
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>'
```

应用程序保护

列出快照

您可以列出为特定受管应用程序创建的快照。

开始之前

您必须具有要列出其快照的受管应用程序的 ID 。如果需要，您可以使用此工作流 "列出受管应用程序" 以查找应用程序。

1. 列出快照

执行以下 REST API 调用。

HTTP 方法	路径
获取	/accouns/ { account_id } /K8s/v1/managedApps/ { managedApp_id } /appSaps

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	Type	Required	Description
受管应用程序 ID	路径	是的。	标识拥有列出快照的受管应用程序。
count	查询	否	如果为 `count=true` ，则快照数包含在响应的元数据部分中。

curl 示例：返回应用程序的所有快照

```
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 示例：返回应用程序和计数的所有快照

```
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 输出示例

```
{
  "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
  }
}
```


列出备份

您可以列出为特定受管应用程序创建的备份。

开始之前

您必须具有要列出备份的受管应用程序的 ID 。如果需要，您可以使用此工作流 "列出受管应用程序" 以查找应用程序。

1. 列出备份

执行以下 REST API 调用。

HTTP 方法	路径
获取	/accouns/ { account_id } /K8s/v1/managedApps/ { managedApp_id } /appBackups

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	Type	Required	Description
受管应用程序 ID	路径	是的。	标识拥有列出备份的受管应用程序。

curl 示例：返回应用程序的所有备份

```
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 输出示例

```
{
  "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": {}
}
```

为受管应用程序创建快照

您可以为特定受管应用程序创建快照。

开始之前

您必须具有要为其创建快照的受管应用程序的 ID 。如果需要，您可以使用此工作流 "列出受管应用程序" 以查找应用程序。

1. 创建快照

执行以下 REST API 调用。

HTTP 方法	路径
发布	/accouns/ { account_id } /K8s/v1/managedApps/ { managedApp_id } /appSaps

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	**Type**	**Required**	**Description**
受管应用程序 ID	路径	是的。	标识要创建快照的受管应用程序。
JSON	body	是的。	提供快照的参数。请参见以下示例。

JSON 输入示例

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

curl 示例：为应用程序创建快照

```
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
```

为受管应用程序创建备份

您可以为特定受管应用程序创建备份。您可以使用备份还原或克隆应用程序。

开始之前

您必须具有要创建备份的受管应用程序的 ID 。如果需要，您可以使用此工作流 "列出受管应用程序" 以查找应用程序。

1. 创建备份

执行以下 REST API 调用。

HTTP 方法	路径
发布	/accouns/ { account_id } /K8s/v1/managedApps/ { managedApp_id } /appBackups

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	Type	Required	Description
受管应用程序 ID	路径	是的。	标识要创建备份的受管应用程序。
JSON	body	是的。	提供备份的参数。请参见以下示例。

JSON 输入示例

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

curl 示例：为应用程序创建备份

```
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
```

删除快照

您可以删除与受管应用程序关联的快照。

开始之前

您必须具备以下条件：

- 拥有快照的受管应用程序的 ID 。如果需要，您可以使用此工作流 "列出受管应用程序" 以查找应用程序。
- 要删除的快照的 ID 。如果需要，您可以使用此工作流 "列出快照" 以查找快照。

1. 删除快照

执行以下 REST API 调用。

HTTP 方法	路径
删除	/accounts- { account_id } /K8s/v1/managedApps/ { managedApp_id } /appSnaps/ { appsnap_id }

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	Type	Required	Description
受管应用程序 ID	路径	是的。	标识拥有快照的受管应用程序。
Snapshot ID	路径	是的。	标识要删除的快照。

curl 示例：删除应用程序的单个快照

```
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>'
```

删除备份

您可以删除与受管应用程序关联的备份。

开始之前

您必须具备以下条件：

- 拥有备份的受管应用程序的 ID 。如果需要，您可以使用此工作流 "列出受管应用程序" 以查找应用程序。
- 要删除的备份的 ID 。如果需要，您可以使用此工作流 "列出备份" 以查找快照。

1. 删除备份

执行以下 REST API 调用。

> 您可以使用可选的请求标头强制删除失败的备份，如下所述。

HTTP 方法	路径
删除	/accouns/ { account_id } /K8s/v1/managedApps/ { managedApp_id } /appBackups/ { appBackup_id }

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	Type	Required	Description
受管应用程序 ID	路径	是的。	标识拥有备份的受管应用程序。
备份 ID	路径	是的。	标识要删除的备份。
强制删除	标题	否	用于强制删除失败的备份。

curl 示例：删除应用程序的单个备份

```
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 示例：使用 **force** 选项删除应用程序的单个备份

```
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'
```

克隆和还原应用程序

克隆受管应用程序

您可以通过克隆现有受管应用程序来创建新应用程序。

开始之前

请注意以下有关此工作流的信息：

- 未使用应用程序备份或快照
- 克隆操作在同一集群中执行

> 要将应用程序克隆到其他集群，您需要根据环境需要更新 JSON 输入中的 `clusterid` 参数。

选择要克隆的受管应用程序

执行工作流 "列出受管应用程序" 并选择要克隆的应用程序。用于克隆应用程序的 REST 调用需要多个资源值。

2. 克隆应用程序

执行以下 REST API 调用。

HTTP 方法	路径
发布	/account/ { accountID } /K8s/v1/managedApps

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	Type	Required	Description
JSON	body	是的。	提供克隆应用程序的参数。请参见以下示例。

JSON 输入示例

```
{
  "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 示例：克隆应用程序

```
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
```

从快照克隆受管应用程序

您可以通过从应用程序快照克隆新应用程序来创建该应用程序。

开始之前

请注意以下有关此工作流的信息：

- 使用应用程序快照
- 克隆操作在同一集群中执行

> 要将应用程序克隆到其他集群，您需要根据环境需要更新 JSON 输入中的 `clusterid` 参数。

选择要克隆的受管应用程序

执行工作流 "列出受管应用程序" 并选择要克隆的应用程序。用于克隆应用程序的 REST 调用需要多个资源值。

2. 选择要使用的快照

执行工作流 "列出快照" 并选择要使用的快照。

3. 克隆应用程序

执行以下 REST API 调用。

HTTP 方法	路径
发布	/account/ { accountID } /K8s/v1/managedApps

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	**Type**	**Required**	**Description**
JSON	body	是的。	提供克隆应用程序的参数。请参见以下示例。

JSON 输入示例

```
{
  "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 示例：从快照克隆应用程序

```
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
```

从备份克隆受管应用程序

您可以通过从应用程序备份克隆新的托管应用程序来创建该应用程序。

开始之前

请注意以下有关此工作流的信息：

- 使用应用程序备份
- 克隆操作在同一集群中执行

> 要将应用程序克隆到其他集群，您需要根据环境需要更新 JSON 输入中的 `clusterid` 参数。

选择要克隆的受管应用程序

执行工作流 "列出受管应用程序" 并选择要克隆的应用程序。用于克隆应用程序的 REST 调用需要多个资源值。

2. 选择要使用的备份

执行工作流 "列出备份" 并选择要使用的备份。

3. 克隆应用程序

执行以下 REST API 调用。

HTTP 方法	路径
发布	/account/ ｛accountID｝ /K8s/v1/managedApps

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	**Type**	**Required**	**Description**
JSON	body	是的。	提供克隆应用程序的参数。请参见以下示例。

JSON 输入示例

```
{
  "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 示例：从备份克隆应用程序

```
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
```

从备份还原受管应用程序

您可以通过从备份创建新应用程序来还原受管应用程序。

1. 选择要还原的受管应用程序

执行工作流 "列出受管应用程序" 并选择要克隆的应用程序。用于克隆应用程序的 REST 调用需要多个资源值。

2. 选择要使用的备份

执行工作流 "列出备份" 并选择要使用的备份。

3. 还原应用程序

执行以下 REST API 调用。您必须提供备份（如下所示）或快照的 ID 。

HTTP 方法	路径
PUT	/account/ { accountID } /K8s/v1/managedApps/ { appID }

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	**Type**	**Required**	**Description**
JSON	body	是的。	提供克隆应用程序的参数。请参见以下示例。

JSON 输入示例

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

curl 示例：从备份原位还原应用程序

```
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
```

支持

列出通知

您可以列出特定 Astra 帐户的通知。您可以在监控系统活动或调试问题描述时执行此操作。

1. 列出通知

执行以下 REST API 调用。

HTTP 方法	路径
获取	/account/ ｛accountID｝ /core/v1/notifications

其他输入参数

除了所有 REST API 调用通用的参数之外，此步骤的 curl 示例还使用以下参数。

参数	**Type**	**Required**	**Description**
筛选器	查询	否	也可以筛选要在响应中返回的通知。
包括	查询	否	也可以选择要在响应中返回的值。

curl 示例：返回所有通知

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

curl 示例：返回严重性为警告的通知的问题描述

```
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 输出示例

```
{
  "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": {}
}
```

删除失败的应用程序

如果某个受管应用程序的备份或快照处于故障状态，您可能无法将其删除。在这种情况下，您可以使用下面所述的工作流手动删除此应用程序。

1. 选择要删除的受管应用程序

执行工作流 "列出受管应用程序" 并选择要删除的应用程序。

2. 列出应用程序的现有备份

执行工作流 "列出备份"。

3. 删除所有备份

通过执行此工作流删除所有应用程序备份 "删除备份" 列表中的每个备份。

4. 列出应用程序的现有快照

执行工作流 "列出快照"。

5. 删除所有快照

执行工作流 "删除快照" 列表中的每个快照。

6. 删除应用程序

执行工作流 "取消管理应用程序" 删除应用程序。

使用 Python

NetApp Astra Control Python SDK

NetApp Astra Control Python SDK 是一个开源软件包，可用于自动部署 Astra Control 。该软件包也是了解 Astra Control REST API 的宝贵资源，或许可以在创建您自己的自动化平台时使用。

> 为了简单起见，本页其余部分将 NetApp Astra Control Python SDK 称为 * SDK * 。

两个相关软件工具

SDK 包含两个不同但相关的工具，这些工具在访问 Astra Control REST API 时在不同的抽象级别运行。

Astra SDK

Astra SDK 提供核心平台功能。它包括一组 Python 类，用于抽象化底层 REST API 调用。这些类支持对各种 Astra Control 资源执行管理操作，包括应用程序，备份，快照和集群。

Astra SDK 是该软件包的一部分，并在单个 `astraSDK.py` 文件中提供。您可以将此文件导入到您的环境中并直接使用这些类。

> NetApp Astra Control Python SDK* （或仅 SDK ）是整个软件包的名称。* Astra SDK* 是指单个文件 `astraSDK.py` 中的核心 Python 类。

工具包脚本

除了 Astra SDK 文件之外，还可以使用 `toolkup.py` 脚本。此脚本可通过访问内部定义为 Python 函数的独立管理操作，在较高的抽象级别下运行。该脚本将导入 Astra SDK 并根据需要调用类。

如何访问

您可以通过以下方式访问 SDK 。

Python 软件包

SDK 可从获取 "Python 软件包索引" 名称为 * NetApp-Astra-toolkits* 。软件包将分配一个版本号，并将根据需要继续更新。您必须使用 * Pip* 软件包管理实用程序将软件包安装到您的环境中。

请参见 "PyPI： NetApp Astra Control Python SDK" 有关详细信息 …

GitHub 源代码

此外，还可以从 GitHub 获取 SDK 源代码。存储库包括以下内容：

- `astraSDK.py` （采用 Python 类的 Astra SDK ）
- `toolkit .py` （基于函数的高级脚本）
- 详细的安装要求和说明
- 安装脚本

- 其他文档

您可以克隆 "GitHub： NetApp/NetApp-Astra-toolkits." 存储库连接到本地环境。

安装和基本要求

在安装软件包并准备使用该软件包时，需要考虑多个选项和要求。

安装选项摘要

您可以通过以下方式之一安装 SDK：

- 使用 Pip 将 PyPI 中的软件包安装到 Python 环境中
- 克隆 Git Hub 存储库，然后执行以下任一操作：
  - 将软件包部署为 Docker 容器（其中包括您需要的所有内容）
  - 复制这两个核心 Python 文件，以便可以通过 Python 客户端代码访问它们

有关详细信息，请参见 PyPI 和 GitHub 页面。

Astra Control 环境的要求

无论是直接使用 Astra SDK 中的 Python 类，还是使用 `toolkape.py` 脚本中的功能，最终您将在部署 Astra Control 时访问 REST API 。因此，您需要一个 Astra 帐户以及一个 API 令牌。请参见 "开始之前" 有关详细信息，请参见本文档 * 入门 * 一节中的其他页面。

NetApp Astra Control Python SDK 的要求

SDK 具有与本地 Python 环境相关的几个前提条件。例如，您必须使用 Python 3.5 或更高版本。此外，还需要几个 Python 软件包。有关详细信息，请参见 GitHub 存储库页面或 PyPI 软件包页面。

有用资源摘要

下面是您开始使用所需的一些资源。

- "PyPI： NetApp Astra Control Python SDK"
- "GitHub： NetApp/NetApp-Astra-toolkits."

原生 Python

开始之前

Python 是一种受欢迎的开发语言，尤其适用于数据中心自动化。在将 Python 的原生功能与多个通用软件包结合使用之前，您需要准备环境和所需的输入文件。

> 除了使用 Python 直接访问 Astra Control REST API 之外， NetApp 还提供了一个工具包软件包，用于抽象化 API 并消除一些复杂性。请参见 "NetApp Astra Control Python SDK" 有关详细信息 …

准备环境

下面介绍了运行 Python 脚本的基本配置要求。

Python 3.

您需要安装最新版本的 Python 3 。

其他库

必须安装 * 请求 * 和 * urllib3* 库。您可以根据环境需要使用 pip 或其他 Python 管理工具。

网络访问

运行脚本的工作站必须能够访问网络并访问 Astra Control 。使用 Astra 控制服务时，您必须连接到互联网，并且能够连接到 `https://astra.netapp.io` 上的服务。

身份信息

您需要一个具有帐户标识符和 API 令牌的有效 Astra 帐户。请参见 "获取 API 令牌" 有关详细信息 …

创建 **JSON** 输入文件

Python 脚本依赖于 JSON 输入文件中包含的配置信息。下面提供了示例文件。

> 您需要根据环境的具体情况更新这些示例。

身份信息

以下文件包含 API 令牌和 Astra 帐户。您需要使用 ` -i` （或 ` -identity` ） CLI 参数将此文件传递到 Python 脚本。

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

列出受管应用程序

您可以使用以下脚本列出 Astra 帐户的受管应用程序。

> 请参见 "开始之前" 所需 JSON 输入文件的示例。

```
#!/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 参考

您可以访问所有 Astra Control REST API 调用的详细信息，包括 HTTP 方法，输入参数和响应。在使用 REST API 开发自动化应用程序时，此完整参考非常有用。

目前， Astra Control 随附了 REST API 参考文档，可在线获取。

开始之前

您需要一个 Astra 控制中心或 Astra 控制服务帐户。

步骤

1. 使用您的帐户凭据登录到 Astra 。

   访问以下站点以获取 Astra 控制服务： "https://astra.netapp.io"

2. 单击页面右上角的图图标并选择 * API access* 。
3. 在页面顶部，单击 * API Documentage* 下显示的 URL 。
4. 如果出现提示，请重新提供您的帐户凭据。

其他资源

您可以访问其他资源来获取帮助并查找有关 NetApp 云服务和支持以及一般 REST 和云概念的更多信息。

Astra

- "Astra Control Center 22.04 文档"

  在客户内部部署的当前版本 Astra Control Center 软件的文档。

- "Astra Control Service 文档"

  公有云中提供的最新版本的 Astra 控制服务软件文档。

- "Astra Trident 文档"

  由 NetApp 维护的开源存储编排程序 Astra Trident 软件最新版本的文档。

- "Astra 系列文档"

  用于访问适用于内部部署和公有云部署的所有 Astra 文档的中央位置。

NetApp 云资源

- "NetApp 云解决方案"

  NetApp 云解决方案的中央站点。

- "NetApp Cloud Central 控制台"

  可登录的 NetApp Cloud Central 服务控制台。

- "NetApp 支持"

  访问故障排除工具，文档和技术支持帮助。

REST 和云概念

- 博士 "Dissertation" 由 Roy Fielding 主讲

  本出版物介绍并建立了 REST 应用程序开发模型。

- "Auth0"

  这是 Astra 服务用于 Web 访问的身份验证和授权平台服务。

- "RFC 编辑器"

Web 和 Internet 标准的权威来源，作为一组编号唯一的 RFC 文档进行维护。

早期版本的 Astra Control Automation 文档

您可以通过以下链接访问先前版本的 Astra Control 的自动化文档。

- "Astra Control Automation 21.12 文档"
- "Astra Control Automation 21.08 文档"

法律声明

法律声明提供对版权声明、商标、专利等的访问。

版权

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

商标

NetApp 、 NetApp 徽标和 NetApp 商标页面上列出的标记是 NetApp 、 Inc. 的商标。其他公司和产品名称可能是其各自所有者的商标。

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

专利

有关 NetApp 拥有的专利的最新列表，请访问：

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

隐私政策

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

Astra Control API 许可证

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