Astra Control Automation documentation

Astra Automation

NetApp
September 16, 2022

This PDF was generated from https://docs.netapp.com/us-en/astra-automation/index.html on September 16, 2022. Always check docs.netapp.com for the latest.

Table of Contents

Astra Control Automation documentation . . . 1
Release notes . . . 2
- About this release . . . 2
- What's new with the Astra Control REST API . . . 2
- Known issues . . . 5

Introduction to the features and benefits . . . 6
Get started . . . 7
- Before you begin . . . 7
- Get an API token . . . 7
- Hello world . . . 8
- Prepare to use the workflows . . . 9
- Basic Kubernetes concepts . . . 11

Core REST implementation . . . 12
- REST web services . . . 12
- Resources and collections . . . 13
- HTTP details . . . 14
- URL format . . . 17

Resources and endpoints . . . 18
- Summary of Astra Control REST resources . . . 18
- Additional resources and endpoints . . . 20

Additional usage considerations . . . 22
- RBAC security . . . 22
- Work with collections . . . 22
- Diagnostics and support . . . 23
- Revoke an API token . . . 23

Infrastructure workflows . . . 25
- Before you begin . . . 25
- Identity and access . . . 25
- LDAP configuration . . . 26
- Buckets . . . 44
- Storage . . . 45
- Clusters . . . 46

Management workflows . . . 48
- Before you begin . . . 48
- App control . . . 49
- App protection . . . 57
- Cloning and restoring an app . . . 64
- Support . . . 69

Using Python . . . 72
- NetApp Astra Control Python SDK . . . 72
- Native Python . . . 73

API reference . . . 80
Additional resources . . . 81

Astra . . . 81
NetApp cloud resources . . . 81
REST and cloud concepts . . . 81
Earlier versions of Astra Control Automation documentation . . . 83
Legal notices . . . 84
Copyright . . . 84
Trademarks . . . 84
Patents . . . 84
Privacy policy . . . 84
Astra Control API license . . . 84

Astra Control Automation documentation

Release notes

About this release

The documentation at this site describes the Astra Control REST API and related automation technologies available with the August 2022 (22.08) release of Astra Control. In particular, this release of the REST API is included with the corresponding 22.08 releases of Astra Control Center and Astra Control Service.

See the following pages and sites for more information about this release as well as previous releases:

- What's new with the Astra Control REST API
- REST resources and endpoints
- Astra Control Center 22.08 documentation
- Astra Control Service 22.08 documentation
- Earlier versions of Astra Automation documentation

Follow us on Twitter @NetAppDoc. Send feedback about documentation by becoming a GitHub contributor or sending an email to doccomments@netapp.com.

What's new with the Astra Control REST API

NetApp periodically updates the Astra Control REST API to bring you new features, enhancements, and bug fixes.

10 August 2022 (22.08)

This release includes an expansion and update of the REST API as well as enhanced security and administrative features.

New and enhanced Astra resources

Three new resources types have been added: **Certificate**, **Group**, and **AppMirror**. In addition, the versions of several existing resources have been updated.

LDAP authentication

You can optionally configure Astra Control Center to integrate with an LDAP server to authenticate selected Astra users. See LDAP configuration for more information.

Enhanced execution hook

Support for execution hooks was added with the Astra Control 21.12 release. In addition to the existing pre-snapshot and post-snapshot execution hooks, you can now configure the following types of execution hooks with the 22.08 release:

- Pre-backup
- Post-backup

- Post-restore

Astra Control now also allows same script to be used for multiple execution hooks.

Application replication using SnapMirror

You can now replicate data and application changes among clusters using NetApp SnapMirror technology. This enhancement can be used to improve your business continuity and recovery capabilities.

Related information

- Astra Control Center: What's new
- Astra Control Service: What's new

26 April 2022 (22.04)

This release includes an expansion and update of the REST API as well as enhanced security and administrative features.

New and enhanced Astra resources

Two new resources types have been added: **Package** and **Upgrade**. In addition, the versions of several existing resources have been upgraded.

Enhanced RBAC with namespace granularity

When binding a role to an associated user, you can limit the namespaces the user has access to. See the **Role Binding API** reference and RBAC security for more information.

Bucket removal

You can remove a bucket when it is no longer needed or is not functioning properly.

Support for Cloud Volumes ONTAP

Cloud Volumes ONTAP is now supported as a storage backend.

Additional product enhancements

There are several additional enhancements to the two Astra Control product implementations, including:

- Generic ingress for Astra Control Center
- Private cluster in AKS
- Support for Kubernetes 1.22
- Support for VMware Tanzu portfolio

See the **What's new** page at the Astra Control Center and Astra Control Service documentation sites.

Related information

- Astra Control Center: What's new
- Astra Control Service: What's new

14 December 2021 (21.12)

This release includes an expansion of the REST API along with a change to the documentation structure to better support the evolution of Astra Control through the future release updates.

Separate Astra Automation documentation for each release of Astra Control

Every release of Astra Control includes a distinct REST API that has been enhanced and tailored to the features of the specific release. The documentation for each release of the Astra Control REST API is now available at its own dedicated web site along with the associated GitHub content repository. The main doc site Astra Control Automation always contains the documentation for the most current release. See Earlier versions of Astra Control Automation documentation for information about prior releases.

Expansion of the REST resource types

The number of REST resource types has continued to expand with an emphasis on execution hooks and storage backends. The new resources include: account, execution hook, hook source, execution hook override, cluster node, managed storage backend, namespace, storage device, and storage node. See Resources for more information.

NetApp Astra Control Python SDK

NetApp Astra Control Python SDK is an open source package that makes it easier to develop automation code for your Astra Control environment. At the core is the Astra SDK which includes a set of classes to abstract the complexity of the REST API calls. There is also a toolkit script to execute specific administrative tasks by wrapping and abstracting the Python classes. See NetApp Astra Control Python SDK for more information.

5 August 2021 (21.08)

This release includes the introduction of a new Astra deployment model and a major expansion of the REST API.

Astra Control Center deployment model

In addition to the existing Astra Control Service offering provided as a public cloud service, this release also includes the Astra Control Center on-premises deployment model. You can install Astra Control Center at your site to manage your local Kubernetes environment. The two Astra Control deployment models share the same REST API, with minor differences noted as needed in the documentation.

Expansion of the REST resource types

The number of resources accessible through the Astra Control REST API has greatly expanded, with many of the new resources providing a foundation for the on-premises Astra Control Center offering. The new resources include: ASUP, entitlement, feature, license, setting, subscription, bucket, cloud, cluster, managed cluster, storage backend, and storage class. See Resources for more information.

Additional endpoints supporting an Astra deployment

In addition to the expanded REST resources, there are several other new API endpoints available to support an Astra Control deployment.

OpenAPI support

The OpenAPI endpoints provide access to the current OpenAPI JSON document and other related resources.

OpenMetrics support

The OpenMetrics endpoints provide access to account metrics through the OpenMetrics resource.

15 April 2021 (21.04)

This release includes the following new features and enhancements.

Introduction of the REST API

The Astra Control REST API is available for use with the Astra Control Service offering. It has been created based on REST technologies and current best practices. The API provides a foundation for the automation of your Astra deployments and includes the following features and benefits.

Resources

There are fourteen REST resource types available.

API token access

Access to the REST API is provided through an API access token which you can generate at the Astra web user interface. The API token provides secure access to the API.

Support for collections

There is a rich set of query parameters which can be used to access the resources collections. Some of the supported operations include filtering, sorting, and pagination.

Known issues

You should review all the known issues for the current release related to the Astra Control REST API. The known issues identify problems that might prevent you from using the product successfully.

There are no new known issues with the 22.08 release of the Astra Control REST API. The issues described below were discovered in previous releases and are still applicable with the current release.

Not all storage devices in a backend storage node are discovered

When issuing a REST API call to retrieve the storage devices defined in a storage node, only the Astra Data Store devices are discovered. Not all the devices are returned.

Astra Data Store storage backend in `Unknown` state

The Astra Data Store storage backend is in the `Unknown` state after issuing an API call to retrieve the storage backend. In this condition, the storage backend is actually still available and can be communicated with. However, a component within the storage backend is likely in an unhealthy state and needs to be returned to a healthy state for the storage backend to show as `Available`.

Introduction to the features and benefits

Astra Control Center and Astra Control Service provide a common REST API that you can access directly through a programming language or utility such as Curl. The major highlights and benefits of the API are presented below.

To access the REST API, you need to first sign in to the Astra web user interface and generate an API token. You must include the token with each API request.

Built on REST technology

The Astra Control API has been created using REST technology and current best practices. The core technology includes HTTP, JSON, and RBAC.

Support for the two Astra Control deployment models

Astra Control Service is used within the public cloud environment while Astra Control Center is for your on-premises deployments. There is one REST API supporting both of these deployment models.

Clear mapping between REST endpoint resources and object model

The external REST endpoints used to access the resources map to a consistent object model maintained internally by the Astra service. The object model is designed using entity-relationship (ER) modeling which helps to clearly define the API actions and responses.

Rich set of query parameters

The REST API provides a rich set of query parameters that you can use to access the resources collections. Some of the supported operations include filtering, sorting, and pagination.

Alignment with the Astra Control web UI

The design of the Astra web user interface is aligned with the REST API and so there is consistency between the two access paths and user experience.

Robust debugging and problem determination data

The Astra Control REST API provides a robust debugging and problem determination capability, including system events and user notifications.

Workflow processes

A set of workflows is provided to assist with the development of your automation code. The workflows are organized in two major categories: infrastructure and management.

Foundation for advanced automation technologies

In addition to accessing the REST API directly, you can use other automation technologies which are based on the REST API.

Part of the Astra family documentation

The Astra Control Automation documentation is part of the larger Astra family documentation. See Astra documentation for more information.

Get started

Before you begin

You can quickly prepare to get started with the Astra Control REST API by reviewing the steps below.

Have Astra account credentials

You'll need Astra credentials to sign in to the Astra web user interface and generate an API token. With Astra Control Center, you manage these credentials locally. With Astra Control Service the account credentials are accessed through the **Auth0** service.

Become familiar with basic Kubernetes concepts

You should be familiar with several basic Kubernetes concepts. See Basic Kubernetes concepts for more information.

Review REST concepts and implementation

Make sure to review Core REST implementation for information about REST concepts and the details regarding how the Astra Control REST API is designed.

Get more information

You should be aware of the additional information resources as suggested in Additional resources.

Get an API token

You need to obtain an Astra API token to use the Astra Control REST API.

Introduction

An API token identifies the caller to Astra and must be included with every REST API call.

- You can generate an API token using the Astra web user interface.
- The user identity carried with the token is determined by the user creating the token.
- The token must be included in the `Authorization` HTTP request header.
- A token never expires after it is created.
- You can revoke a token at the Astra web user interface.

Related information

- Revoke an API token

Create an Astra API token

The following steps describe how to create an Astra API token.

Before you begin

You need credentials for an Astra account.

About this task

This task generates an API token at the Astra web interface. You should also retrieve the account ID which is also needed when making an API calls.

Steps

1. Sign in to Astra using your account credentials.

   Access the following site for Astra Control Service: https://astra.netapp.io

2. Click the figure icon at the top right of the page and select **API access**.
3. Click **Generate API token** on the page and in the popup window click **Generate API token**.
4. Click the icon to copy the token string to the clipboard and save it in your editor.
5. Copy and save the account id which is available on the same page.

After you finish

When you access the Astra Control REST API through Curl or a programming language, you must include the API bearer token in the HTTP `Authorization` request header.

Hello world

You can issue a simple curl command at your workstation's CLI to get started using the Astra Control REST API and confirm its availability.

Before you begin

The Curl utility must be available on your local workstation. You must also have an API token and the associated account identifier. See Get an API token for more information.

Curl example

The following Curl command retrieves a list of Astra users. Provide the appropriate <ACCOUNT_ID> and <API_TOKEN> as indicated.

```
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 output example

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

Prepare to use the workflows

You should be familiar with the organization and format of the Astra workflows before using them with a live deployment.

Introduction

A *workflow* is a sequence of one or more steps needed to accomplish a specific administrative task or goal. Each step in an Astra Control workflow is one of the following:

- REST API call (with details such as curl and JSON examples)
- Invocation of another Astra workflow
- Miscellaneous related task (such as making a required design decision)

The workflows include the core steps and parameters needed to accomplish each task. They provide a starting point for customizing your automation environment.

Common input parameters

The input parameters described below are common to all the curl samples used to illustrate a REST API call.

Because these input parameters are universally required, they are not described further in the individual workflows. If additional input parameters are used for a specific curl example, they are described in the section **Additional input parameters**.

Path parameters

The endpoint path used with every REST API call includes the following parameters. Also see URL format for more information.

Account ID

This is the UUIDv4 value identifying the Astra account where the API operation runs. See Get an API token for more information about locating your account ID.

Request headers

There are several request headers that you may need to include depending on the REST API call.

Authorization

All the API calls in the workflows need an API token to identify the user. You must include the token in the `Authorization` request header. See Get an API token for more information about generating an API token.

Content type

With the HTTP POST and PUT requests where JSON is included in the request body, you should declare the media type based on the Astra resource. For example, you can include the header `Content-Type: application/astra-appSnap+json` when creating a snapshot for a managed application.

Accept

You can declare the specific media type of the content you expect in the response based on the Astra resource. For example, you can include the header `Accept: application/astra-appBackup+json` when listing the backups for a managed application. However, for simplicity the curl samples in the workflows accept all media types.

Presentation of tokens and identifiers

The API token and other ID values used with the curl examples are opaque with no discernible meaning. And so to improve the readability of the samples, the actual token and ID values are not used. Rather, smaller reserved keywords are used which has several benefits:

- The curl and JSON samples are clearer and easier to understand.
- Because all the keywords use the same format with brackets and capital letters, you can quickly identify the location and content to insert or extract.
- No value is lost because the original parameters cannot be copied and used with an actual deployment.

Here are some of the common reserved keywords used in the curl examples. This list is not exhaustive and additional keywords are used as needed. Their meaning should be obvious based on the context.

Keyword	Type	Description
<ACCOUNT_ID>	Path	The UUIDv4 value identifying the account where the API operation runs.
<API_TOKEN>	Header	The bearer token identifying and authorizing the caller.
<MANAGED_APP_ID>	Path	The UUIDv4 value identifying the managed application for the API call.

Workflow categories

There are two broad categories of Astra workflows available based on your deployment model. If you are using Astra Control Center, you should start with the infrastructure workflows and then proceed to the management workflows. When using Astra Control Service, you can typically go directly to the management workflows.

The curl samples in the workflows use the URL for the Astra Control Service. You need to change the URL when using the on-premises Astra Control Center as appropriate for your environment.

Infrastructure workflows

These workflows deal with the Astra infrastructure, including credentials, buckets, and storage backends. They are needed with Astra Control Center but in most cases can also be used with Astra Control Service. The workflows focus on the tasks required to establish and maintain an Astra managed cluster.

Management workflows

You can use these workflows after you have a managed cluster. The workflows focus on application protection and support operations such as backing up, restoring, and cloning a managed app.

Basic Kubernetes concepts

There are several Kubernetes concepts that are relevant when using the Astra REST API.

Objects

The objects maintained within a Kubernetes environment are persistent entities representing the configuration of the cluster. These objects collectively describe the state of the system including the cluster workload.

Namespaces

Namespaces provide a technique for isolating resources within a single cluster. This organizational structure is useful when dividing the types of work, users, and resources. Objects with a *namespace scope* need to be unique within the namespace, while those with a *cluster scope* must be unique across the entire cluster.

Labels

Labels can be associated with the Kubernetes objects. They describe attributes using key-value pairs and can enforce an arbitrary organization on the cluster which can be useful to an organization but are outside the core Kubernetes operation.

Core REST implementation

REST web services

Representational State Transfer (REST) is a style for creating distributed web applications. When applied to the design of a web services API, it establishes a set of mainstream technologies and best practices for exposing server-based resources and managing their states. While REST provides a consistent foundation for application development, the details of each API can vary based on the specific design choices. You should be aware of the characteristics of the Astra Control REST API before using it with a live deployment.

Resources and state representation

Resources are the basic components of a web-based system. When creating a REST web services application, early design tasks include:

- Identification of system or server-based resources

  Every system uses and maintains resources. A resource can be a file, business transaction, process, or administrative entity. One of the first tasks in designing an application based on REST web services is to identify the resources.

- Definition of resource states and associated state operations

  Resources are always in one of a finite number of states. The states, as well as the associated operations used to affect the state changes, must be clearly defined.

URI endpoints

Every REST resource must be defined and made available using a well-defined addressing scheme. The endpoints where the resources are located and identified use a Uniform Resource Identifier (URI). The URI provides a general framework for creating a unique name for each resource in the network. The Uniform Resource Locator (URL) is a type of URI used with web services to identify and access resources. Resources are typically exposed in a hierarchical structure similar to a file directory.

HTTP messages

Hypertext Transfer Protocol (HTTP) is the protocol used by the web services client and server to exchange request and response messages about the resources. As part of designing a web services application, HTTP methods are mapped to the resources and corresponding state management actions. HTTP is stateless. Therefore, to associate a set of related requests and responses as part of one transaction, additional information must be included in the HTTP headers carried with the request and response data flows.

JSON formatting

While information can be structured and transferred between a web services client and server in several ways, the most popular option is JavaScript Object Notation (JSON). JSON is an industry standard for representing simple data structures in plain text and is used to transfer state information describing the resources. The Astra Control REST API uses JSON to format the data carried in the body of each HTTP request and response.

Resources and collections

The Astra Control REST API provides access to resource instances and collections of resource instances.

Conceptually a REST **resource** is similar to an **object** as defined with the object-oriented programming (OOP) languages and systems. Sometimes these terms are used interchangeably. But in general, "resource" is preferred when used in the context of the external REST API while "object" is used for the corresponding stateful instance data stored at the server.

Attributes of the Astra resources

The Astra Control REST API conforms to RESTful design principles. Each Astra resource instance is created based on a well-defined resource type. A set of resource instances of the same type is referred to as a **collection**. The API calls act on individual resources or collections of resources.

Resource types

The resource types included with the Astra Control REST API have the following characteristics:

- Every resource type is defined using a schema (typically in JSON)
- Every resource schema includes the resource type and version
- Resource types are globally unique

Resource instances

Resource instances available through the Astra Control REST API have the following characteristics:

- Resource instances are created based on a single resource type
- The resource type is indicated using the Media Type value
- Instances are composed of stateful data which is maintained by the Astra service
- Each instance is accessible through a unique and long-lived URL
- In cases where a resource instance can have more than one representation, different media types can be used to request the desired representation

Resource collections

Resource collections available through the Astra Control REST API have the following characteristics:

- The set of resource instances of a single resource type is known as a collection
- Collections of resources have a unique and long-lived URL

Instance identifiers

Every resource instance is assigned an identifier when it is created. This identifier is a 128-bit UUIDv4 value. The assigned UUIDv4 values are globally unique and immutable. After issuing an API call that creates a new instance, a URL with the associated id is returned to the caller in a `Location` header of the HTTP response. You can extract the identifier and use it on subsequent calls when referring to the resource instance.

The resource identifier is the primary key used for collections.

Common structure for Astra resources

Every Astra Control resource is defined using a common structure.

Common data

Every Astra resource contains the key-values shown in the following table.

Key	Description
type	A globally unique resource type which is known as the **resource type**.
version	A version identifier which is known as the **resource version**.
id	A globally unique identifier which is known as the **resource identifier**.
metadata	A JSON object containing various information, including user and system labels.

Metadata object

The metadata JSON object included with each Astra resource contains the key-values shown in the following table.

Key	Description
labels	JSON array of client-specified labels associated with the resource.
creationTimestamp	JSON string containing a timestamp indicating when the resource was created.
modificationTimestamp	JSON string containing an ISO-8601 formatted timestamp indicating when the resource was last altered.
createdBy	JSON string containing the UUIDv4 identifier of the user id that created the resource. If the resource was created by an internal system component and there is no UUID associated with the creating entity, the **null** UUID is used.

Resource state

Selected resources a `state` value which is used to orchestrate lifecycle transitions and control access.

HTTP details

The Astra Control REST API uses HTTP and related parameters to act on the resources and collections. Details of the HTTP implementation are presented below.

API transactions and the CRUD model

The Astra Control REST API implements a transactional model with well-defined operations and state transitions.

Request and response API transaction

Every REST API call is performed as an HTTP request to the Astra service. Each request generates an associated response back to the client. This request-response pair can be considered an API transaction.

Support for CRUD operational model

Each of the resource instances and collections available through the Astra Control REST API is accessed

based on the **CRUD** model. There are four operations, each of which maps to a single HTTP method. The operations include:

- Create
- Read
- Update
- Delete

For some of the Astra resources, only a subset of these operations is supported. You should review the API reference for more information about a specific API call.

HTTP methods

The HTTP methods or verbs supported by the API are presented in the table below.

Method	CRUD	Description
GET	Read	Retrieves object properties for a resource instance or collection. This is considered a **list** operation when used with a collection.
POST	Create	Creates a new resource instance based on the input parameters. The long-term URL is returned in a `Location` response header.
PUT	Update	Updates an entire resource instance with the supplied JSON request body. Key values that are not user modifiable are preserved.
DELETE	Delete	Deletes an existing resource instance.

Request and response headers

The following table summaries the HTTP headers used with the Astra Control REST API.

See RFC 7232 and RFC 7233 for more information.

Header	Type	Usage notes
Accept	Request	If the value is "/" or is not provided, `application/json` is returned in Content-Type response header. If the value is set to the Astra resource Media Type, the same Media Type is returned in the Content-Type header.
Authorization	Request	Bearer token with the API key for the user.
Content-Type	Response	Returned based on the `Accept` request header.
Etag	Response	Included with a successful as defined with RFC 7232. The value is a hexadecimal representation of the MD5 value for the entire JSON resource.
If-Match	Request	A precondition request header implemented as described in section 3.1 RFC 7232 and support for **PUT** requests.
If-Modified-Since	Request	A precondition request header implemented as described in section 3.4 RFC 7232 and support for **PUT** requests.

Header	Type	Usage notes
If-Unmodified-Since	Request	A precondition request header implemented as described in section 3.4 RFC 7232 and support for **PUT** requests.
Location	Response	Contains the full URL of the newly created resource.

Query parameters

The following query parameters are available for use with resource collections. See Working with collections for more information.

Query parameter	Description
include	Contains the fields that should be returned when reading a collection.
filter	Indicates the fields that must match for a resource to be returned when reading a collection.
orderBy	Determines the sort order of resources returned when reading a collection.
limit	Limits the maximum number of resources returned when reading a collection.
skip	Sets the number of resources to pass over and skip when reading a collection.
count	Indicates if the total number of resources should be returned in the metadata object.

HTTP status codes

The HTTP status codes used by the Astra Control REST API are described below.

The Astra Control REST API also uses the **Problem Details for HTTP APIs** standard. See Diagnostics and support for more information.

Code	Meaning	Description
200	OK	Indicates success for calls that do not create a new resource instance.
201	Created	An object is successfully created and the location response header includes the unique identifier for the object.
204	No content	The request was successful although no content was returned.
400	Bad request	The request input is not recognized or is inappropriate.
401	Unauthorized	The user is not authorized and must authentiate.
403	Forbidden	Access is denied due to an authorization error.
404	Not found	The resource referred to in the request does not exist.
409	Conflict	An attempt to create an object failed because the object already exists.
500	Internal error	A general internal error occurred at the server.
503	Service unavailable	The service is not ready to handle the request for some reason.

URL format

The general structure of the URL used to access a resource instance or collection through the REST API is composed of several values. This structure reflects the underlying object model and system design.

Account as the root

The root of the resource path to every REST endpoint is the Astra account. And so all paths in the URL begin with `/account/{account_id}` where `account_id` is the unique UUIDv4 value for the account. Internally structure this reflects a design where all resource access is based on a specific account.

Endpoint resource category

The Astra resource endpoints fall into three different categories:

- Core (`/core`)
- Managed application (`/k8s`)
- Topology (`/topology`)

See Resources for more information.

Category version

Each of the three resource categories has a global version that controls the version of the resources accessed. By convention and definition, moving to a new major version of a resource category (such as, from `/v1` to `/v2`) will introduce breaking changes in the API.

Resource instance or collection

A combination of resource types and identifiers can be used in the path, based on whether a resource instance or collection is accessed.

Example

- Resource path

  Based on the structure presented above, a typical path to an endpoint is: `/accounts/{account_id}/core/v1/users`.

- Complete URL

  The full URL for the corresponding endpoint is: `https://astra.netapp.io/accounts/{account_id}/core/v1/users`.

Resources and endpoints

You can access the resources provided through the Astra Control REST API to automate an Astra deployment. Each resource is available through one or more endpoints. The information presented below provides an introduction to the REST resources you can use as part of an automation deployment.

The format of the path and full URL used to access the Astra Control resources is based on several values. See URL format for more information. Also see API reference for more details about using the Astra resources and endpoints.

Summary of Astra Control REST resources

The primary resource endpoints provided in the Astra Control REST API are organized in three categories. Each resource can be accessed with the full set of CRUD operations (create, read, update, delete) except where noted.

The **Release** column indicates the Astra release when the resource was first introduced. This field is bolded for resources newly added with the current release.

Core resources

The core resource endpoints provide the foundational services needed to establish and maintain the Astra runtime environment.

Resource	Release	Description
Account	21.12	The account resources allow you to manage the isolated tenants within the multitenant Astra Control deployment environment.
ASUP	21.08	The ASUP resources represent the AutoSupport bundles forwarded to NetApp support.
Certificate	**22.08**	The certificate resources represent the installed certificates used for strong authentication for outgoing connections.
Credential	21.04	The credential resources contain security related information which can be used with Astra users, clusters, buckets, and storage backends.
Entitlement	21.08	The entitlement resources represent the features and capacities available for an account based on the active licenses and subscriptions.
Event	21.04	The event resources represent all the events occurring in the system, including the subset classified as notifications.
Execution hook	21.12	The execution hook resources represent custom scripts that you can run either before or after a snapshot of a managed app is performed.
Feature	21.08	The feature resources represent selected Astra features that you can query to determine if they are enabled or disabled in the system. Access is limited to read-only.
Group	**22.08**	The group resources represent the Astra groups and associated resources. Only LDAP groups are supported in the current release.

Resource	Release	Description
Hook source	21.12	The hook source resources represent the actual source code used with an execution hook. Separating the source code from the execution control has several benefits such as allowing the scripts to be shared.
License	21.08	The license resources represent the licenses available for an Astra account.
Notification	21.04	The notification resources represent Astra events that have a notification destination. Access is provided on a per-user basis.
Package	22.04	The package resources provide registration of and access to package definitions. Software packages consist of various components including files, images, and other artifacts.
Role binding	21.04	The role binding resources represent the relationships between specific pairs of users and accounts. In addition to the linkage between the two, a set of permissions is specified for each through a specific role.
Setting	21.08	The setting resources represent a collection of key-value pairs which describe a feature for a specific Astra account.
Subscription	21.08	The subscription resources represent the active subscriptions for an Astra account.
Token	21.04	The token resources represent the tokens available to programmatically access the Astra Control REST API.
Unread notification	21.04	The unread notification resources represent notifications assigned to a specific user but not yet read.
Upgrade	22.04	The upgrade resources provide access to software components and the ability to initiate upgrades.
User	21.04	The user resources represent Astra users able to access the system based on their defined role.

Managed application resources

The managed application resource endpoints provide access to the managed Kubernetes applications.

Resource	Release	Description
Application asset	21.04	The application asset resources represent internal collections of state information needed to manage the Astra applications.
Application backup	21.04	The application backup resources represent backups of the managed applications.
Application snapshot	21.04	The application snapshot resources represent snapshots of the managed applications.
Execution hook override	21.12	The execution hook override resources allow you to disable the preloaded NetApp default execution hooks for specific applications as needed.
Managed application	21.04	The managed app resources represent Kubernetes applications that are managed by Astra.
Schedule	21.04	The schedule resources represent data protection operations that are scheduled for the managed applications as part of a data protection policy.

Topology resources

The topology resource endpoints provide access to the unmanaged applications and storage resources.

Resource	Release	Description
App	21.04	The app resources represent all of the Kubernetes applications, including those unmanaged by Astra.
AppMirror	**22.08**	The AppMirror resources represent the AppMirror resources to provide for the management of application mirroring relationships.
Bucket	21.08	The bucket resources represent the S3 cloud buckets used to store backups of the applications managed by Astra.
Cloud	21.08	The cloud resources represent clouds that Astra clients can connect to in order to manage clusters and applications.
Cluster	21.08	The cluster resources represent the Kubernetes clusters not managed by Kubernetes.
Cluster node	21.12	The cluster node resources provide additional resolution by allowing you to access the individual nodes within a Kubernetes cluster.
Managed cluster	21.08	The managed cluster resources represent the Kubernetes clusters currently managed by Kubernetes.
Managed storage backend	21.12	The managed storage backend resources allow you to access abstracted representations of the backend storage providers. These storage backends can be used by the managed clusters and applications.
Namespace	21.12	The namespace resources provide access to the namespaces used within a Kubernetes cluster.
Storage backend	21.08	The storage backend resources represent providers of storage services that can be used by the Astra managed clusters and applications.
Storage class	21.08	The storage class resources represent different classes or types of storage discovered and available to a specific managed cluster.
Storage device	21.12	The storage device resources provide access to the disks associated with a specific storage node for Astra Data Store (ADS) type storage backends. An ADS storage backends is deployed as a Kubernetes clusters.
Storage node	21.12	The storage node resources represent the nodes that are part of an ADS cluster.
Volume	21.04	The volume resources represent the Kubernetes storage volumes associated with the managed applications.

Additional resources and endpoints

There are several additional resources and endpoints that you can use to support an Astra deployment.

These resources and endpoints are not currently included with the Astra Control REST API reference documentation.

OpenAPI

The OpenAPI endpoints provide access to the current OpenAPI JSON document and other related resources.

OpenMetrics

The OpenMetrics endpoints provide access to the account metrics through the OpenMetrics resource. Support is available with the Astra Control Center deployment model.

Additional usage considerations

RBAC security

The Astra REST API supports role-based access control (RBAC) to restrict access to the system functions.

Astra roles

Every Astra user is assigned to a single role which determines the actions that can be performed. The roles are arranged in a hierarchy as described in the table below.

Role	Description
Owner	Has all the permissions of the Admin role and can also delete Astra accounts.
Admin	Has all the permissions of the Member role and can also invite users to join an account.
Member	Can fully manage the Astra application and compute resources.
Viewer	Restricted to only viewing resources.

Enhanced RBAC with namespace granularity

This feature was introduced with the 22.04 release of the Astra REST API.

When a role binding is established for a specific user, a constraint can be applied to limit the namespaces the user has access to. There are several ways this constraint can be defined as described in the table below. See the parameter `roleContraints` in the Role Binding API for more information.

Namespaces	Description
All	The user can access all the namespaces through the wildcard parameter "*"". This is the default value to maintain backwards compatibility.
None	The constraint list is specified although it is empty. This indicates the user cannot access any namespace.
Namespace list	The UUID of a namespace is included which restricts the user to the single namespace. A comma separated list can also be used to allow access to multiple namespaces.
Label	A label is specified and access is allowed to all the matching namespaces.

Work with collections

The Astra Control REST API provides several different ways to access resource collections through the defined query parameters.

Selecting values

You can specify which key-value pairs should be returned for each resource instance using the `include` parameter. All of the instances are returned in the response body.

Filtering

Collection resource filtering allows an API user to specify conditions which determine if a resource is returned in the response body. The `filter` parameter is used to indicate the filtering condition.

Sorting

Collection resource sorting allows an API user to specify the order in which resources are returned in the response body. The `orderBy` parameter is used to indicate the filtering condition.

Pagination

You can enforce pagination by restricting the number of resource instances returned on a request using the `limit` parameter.

Count

If you include the Boolean parameter `count` set to `true`, the number of resources in the returned array for a given response is provided in the metadata section.

Diagnostics and support

There are several support features available with the Astra Control REST API that can be used for diagnostics and debugging.

API resources

There are several Astra features exposed through API resources that provide diagnostic information and support.

Type	Description
Event	System activities that are recorded as part of Astra processing.
Notification	A subset of the Events that are considered important enough to be presented to the user.
Unread notification	The notifications that have yet to be read or retrieved by the user.

Revoke an API token

You can revoke an API token at the Astra web interface when it is no longer needed.

Before you begin

You need an Astra account. You should also identify the tokens you want to revoke.

About this task

After a token is revoked, it is immediately and permanently unusable.

Steps

1. Sign in to Astra using your account credentials.

   Access the following site for Astra Control Service: https://astra.netapp.io

2. Click the figure icon at the top right of the page and select **API access**.

3. Select the token or tokens you want to revoke.
4. Under the **Actions** drop-down box, click **Revoke tokens**.

Infrastructure workflows

Before you begin

You can use these workflows to create and maintain the infrastructure used with the Astra Control Center deployment model. In most case, the workflows can also be used with Astra Control Service.

These workflows can be expanded and enhanced by NetApp at any time and so you should review them periodically.

General preparation

Before using any of the Astra workflows, make sure to review Prepare to use the workflows.

Workflow categories

The infrastructure workflows are organized in different categories to make it easier to locate the one you want.

Category	Description
Identity and access	These workflows allow you to manage identity and how Astra is accessed. The resources include users, credentials, and tokens.
LDAP configuration	You can optionally configure Astra Control Center to use LDAP to authenticate selected users.
Buckets	You can use these workflows to create and manage the S3 buckets used to store backups.
Storage	These workflows allow you to add and maintain storage backends and volumes.
Clusters	You can add managed Kubernetes clusters which allows you to protect and support the applications they contain.

Identity and access

List users

You can list the users that are defined for a specific Astra account.

1. List the users

Perform the following REST API call.

HTTP method	Path
GET	/account/{account_id}/core/v1/users

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
include	Query	No	Optionally select the values you want returned in the response.

Curl example: Return all data for all users

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

Curl example: Return the first name, last name, and id for all users

```
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 output example


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


LDAP configuration

Prepare for LDAP configuration

You can optionally integrate Astra Control Center with a Lightweight Directory Access Protocol (LDAP) server to perform authentication for selected Astra users. LDAP is an industry standard protocol for accessing distributed directory information and a popular choice for enterprise authentication.

Related information

- LDAP Technical Specification Road Map
- LDAP version 3

Overview of the implementation process

At a high level, there are several steps you need to perform to configure an LDAP server to provide authentication for Astra users.

> While the steps presented below are in a sequence, in some cases you can perform them in a different order. For example, you can define the Astra users and groups before configuring the LDAP server.

1. Review Requirements and limitations to understand the options, requirements, and limitations.
2. Select an LDAP server and the desired configuration options (including security).
3. Perform the workflow Configure Astra to use an LDAP server to integrate Astra with the LDAP server.
4. Review the users and groups at the LDAP server to make sure they are defined properly.
5. Perform the appropriate workflow in Add LDAP entries to Astra to identify the users to be authenticated using LDAP.

Requirements and limitations

You should review the Astra configuration essentials presented below, including limitations and configuration options, before configuring Astra to use LDAP for authentication.

Only supported with Astra Control Center

The Astra Control platform provides two deployment models. LDAP authentication is only supported with Astra Control Center deployments.

REST API configuration only

The current release of Astra Control Center only supports configuration of LDAP authentication using the Astra Control REST API. An important aspect of this limitation is the LDAP users are not displayed in the users tab of the Astra web interface. They are available through the REST API at the endpoint `../core/v1/users`.

LDAP server required

You must have an LDAP server to accept and process the Astra authentication requests. Microsoft's Active Directory is supported with the current Astra Control Center release.

Secure connection to the LDAP server

When configuring the LDAP server in Astra, you can optionally define a secure connection. In this case a certificate is needed for the LDAPS protocol.

Configure users or groups

You need to select the users to be authenticated using LDAP. You can do this either by identifying the individual users or a group of users. The accounts must be defined at the LDAP server. They also need to be identified in Astra (type LDAP) which allows the authentication requests to be forwarded to LDAP.

Role constraint when binding a user or group

With the current release of Astra Control Center, the only supported value for `roleConstraint` is "*". This indicates the user is not restricted to a limited set of namespaces and can access all of them. See Add LDAP entries to Astra for more information.

LDAP credentials

The credentials used by LDAP include the username (email address) and the associated password.

Unique email addresses

All email addresses acting as usernames in an Astra Control Center deployment must be unique. You cannot add an LDAP user with an email address that is already defined to Astra. If a duplicate email exists, you need to first delete it from Astra. See Remove users at the Astra Control Center documentation site for more information.

Optionally define LDAP users and groups first

You can add the LDAP users and groups to Astra Control Center even if they don't yet exist in LDAP or if the LDAP server is not configured. This allows you to preconfigure the users and groups before configuring the LDAP server.

A user defined in multiple LDAP groups

If an LDAP user belongs to multiple LDAP groups and the groups have been assigned different roles in Astra, the user's effective role when authenticating will be the most privileged. For example, if a user is assigned the `viewer` role with group1 but has the `member` role in group2, the user's role would be `member`. This is based on the hierarchy used by Astra (highest to lowest):

- Owner
- Admin
- Member
- Viewer

Periodic account synchronization

Astra synchronizes it's users and groups with the LDAP server approximately every 60 second. So if a user or group is added to or removed from LDAP, it can take up to one minute before it is available in Astra.

Disabling and resetting the LDAP configuration

Before attempting to reset the LDAP configuration, you must first disable LDAP authentication. Also, to change the LDAP server (`connectionHost`), you need to perform both operations. See Disable and reset LDAP for more information.

REST API parameters

The LDAP configuration workflows make REST API calls to accomplish the specific tasks. Each API call can include input parameters as shown in the provided samples. See API reference for information about how to locate the reference documentation.

Configure Astra to use an LDAP server

You need to select an LDAP server and configure Astra to use the server as an

authentication provider. The configuration task consists of the steps described below. Each step includes a single REST API call.

1. Add a CA certificate

Perform the following REST API call to add a CA certificate to Astra.

> This step is optional and only required if you want Astra and the LDAP to communicate over a secure channel using LDAPS.

HTTP method	Path
POST	/account/{account_id}/core/v1/certificates

JSON input example

```
{
  "type": "application/astra-certificate",
  "version": "1.0",
  "certUse": "rootCA",
  "cert": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMyVEN",
  "isSelfSigned": "true"
}
```

Note the following about the input parameters:

- `cert` is a JSON string containing a base64 encoded PKCS-11 formatted certificate (PEM encoded).
- `isSelfSigned` should be set to `true` if the certificate is self-signed. The default is `false`.

Curl example

```
curl --location -i --request POST --data @JSONinput
'https://astra.example.com/accounts/<ACCOUNT_ID>/core/v1/certificates'
--header 'Content-Type: application/astra-certificate+json' --header
'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

JSON response example

```
{
  "type": "application/astra-certificate",
  "version": "1.0",
  "id": "a5212e7e-402b-4cff-bba0-63f3c6505199",
  "certUse": "rootCA",
  "cert": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMyVEN",
  "cn": "adldap.example.com",
  "expiryTimestamp": "2023-07-08T20:22:07Z",
  "isSelfSigned": "true",
  "trustState": "trusted",
  "trustStateTransitions": [
  {
    "from": "untrusted",
    "to": [
      "trusted",
      "expired"
    ]
  },
  {
    "from": "trusted",
    "to": [
      "untrusted",
      "expired"
    ]
  },
  {
    "from": "expired",
    "to": [
     "untrusted",
     "trusted"
    ]
  }
  ],
  "trustStateDesired": "trusted",
  "trustStateDetails": [],
  "metadata": {
    "creationTimestamp": "2022-07-21T04:16:06Z",
    "modificationTimestamp": "2022-07-21T04:16:06Z",
    "createdBy": "8a02d2b8-a69d-4064-827f-36851b3e1e6e",
    "modifiedBy": "8a02d2b8-a69d-4064-827f-36851b3e1e6e",
    "labels": []
    }
}
```

2. Add the bind credentials

Perform the following REST API call to add the bind credentials.

HTTP method	Path
POST	/account/{account_id}/core/v1/credentials

JSON input example

```
{
  "name": "ldapBindCredential",
  "type": "application/astra-credential",
  "version": "1.1",
  "keyStore": {
    "bindDn": "dWlkPWFkbWluLG91PXN5c3RlbQ==",
    "password": "cGFzc3dvcmQ="
  }
}
```

Note the following about the input parameters:

- `bindDn` and `password` are the base64 encoded bind credentials of the LDAP admin user that is able to connect and search the LDAP directory. `bindDn` is the LDAP user's email address.

Curl example

```
curl --location -i --request POST --data @JSONinput
'https://astra.example.com/accounts/<ACCOUNT_ID>/core/v1/credentials'
--header 'Content-Type: application/astra-credential+json' --header
'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

JSON response example

```
{
    "type": "application/astra-credential",
    "version": "1.1",
    "id": "3bd9c8a7-f5a4-4c44-b778-90a85fc7d154",
    "name": "ldapBindCredential",
    "metadata": {
        "creationTimestamp": "2022-07-21T06:53:11Z",
        "modificationTimestamp": "2022-07-21T06:53:11Z",
        "createdBy": "527329f2-662c-41c0-ada9-2f428f14c137"
    }
}
```

Note the following the response parameters:

- The `id` of the credential is used in subsequent workflow steps.

3. Retrieve the UUID of the LDAP setting

Perform the following REST API call to retrieve the UUID of the `astra.account.ldap` setting that is included with Astra Control Center.

> The curl example below uses a query parameter to filter the settings collection. You can instead remove the filter to get all the settings and then search for `astra.account.ldap`.

HTTP method	Path
GET	/account/{account_id}/core/v1/settings

Curl example

```
curl --location -i --request GET
'https://astra.example.com/accounts/<ACCOUNT_ID>/core/v1/settings?filter=n
ame%20eq%20'astra.account.ldap'&include=name,id' --header 'Accept: */*'
--header 'Authorization: Bearer <API_TOKEN>'
```

JSON response example

```
{
  "items": [
    ["astra.account.ldap",
    "12072b56-e939-45ec-974d-2dd83b7815df"
    ]
  ],
  "metadata": {}
}
```

4. Update the LDAP setting

Perform the following REST API call to update the LDAP setting and complete the configuration. Use the `id` value from the previous API call for the `<SETTING_ID>` value in the URL path below.

> You can issue a GET request for the specific setting first to see the configSchema. This will provide more information about the required fields in the configuration.

HTTP method	Path
PUT	/account/{account_id}/core/v1/settings/{setting_id}

JSON input example

```
{
  "type": "application/astra-setting",
  "version": "1.0",
  "desiredConfig": {
    "connectionHost": "myldap.example.com",
    "credentialId": "3bd9c8a7-f5a4-4c44-b778-90a85fc7d154",
    "groupBaseDN": "OU=groups,OU=astra,DC=example,DC=com",
    "isEnabled": "true",
    "port": 686,
    "secureMode": "LDAPS",
    "userBaseDN": "OU=users,OU=astra,DC=example,dc=com",
    "userSearchFilter": "((objectClass=User))",
    "vendor": "Active Directory"
    }
}
```

Note the following about the input parameters:

- `isEnabled` should be set to `true` or an error may occur.
- `credentialId` is the id of the bind credential created earlier.
- `secureMode` should be set to `LDAP` or `LDAPS` based on your configuration in the earlier step.
- Only 'Active Directory' is supported as a vendor.

Curl example

```
curl --location -i --request PUT --data @JSONinput
'https://astra.example.com/accounts/<ACCOUNT_ID>/core/v1/settings/<SETTING
_ID>' --header 'Content-Type: application/astra-setting+json' --header
'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

If the call is successful, the HTTP 204 response is returned.

5. Retrieve the LDAP setting

You can optionally perform the following REST API call to retrieve the LDAP settings and confirm the update.

HTTP method	Path
GET	/account/{account_id}/core/v1/settings/{setting_id}

Curl example

```
curl --location -i --request GET
'https://astra.example.com/accounts/<ACCOUNT_ID>/core/v1/settings/<SETTING
_ID>' --header 'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

JSON response example


```
{
  "items": [
  {
    "type": "application/astra-setting",
    "version": "1.0",
    "metadata": {
      "creationTimestamp": "2022-06-17T21:16:31Z",
      "modificationTimestamp": "2022-07-21T07:12:20Z",
      "labels": [],
      "createdBy": "system",
      "modifiedBy": "00000000-0000-0000-0000-000000000000"
    },
    "id": "12072b56-e939-45ec-974d-2dd83b7815df",
    "name": "astra.account.ldap",
    "desiredConfig": {
      "connectionHost": "10.193.61.88",
      "credentialId": "3bd9c8a7-f5a4-4c44-b778-90a85fc7d154",
      "groupBaseDN": "ou=groups,ou=astra,dc=example,dc=com",
      "isEnabled": "true",
      "port": 686,
      "secureMode": "LDAPS",
      "userBaseDN": "ou=users,ou=astra,dc=example,dc=com",
      "userSearchFilter": "((objectClass=User))",
      "vendor": "Active Directory"
    },
    "currentConfig": {
      "connectionHost": "10.193.160.209",
      "credentialId": "3bd9c8a7-f5a4-4c44-b778-90a85fc7d154",
      "groupBaseDN": "ou=groups,ou=astra,dc=example,dc=com",
      "isEnabled": "true",
      "port": 686,
      "secureMode": "LDAPS",
      "userBaseDN": "ou=users,ou=astra,dc=example,dc=com",
      "userSearchFilter": "((objectClass=User))",
      "vendor": "Active Directory"
    },
    "configSchema": {
      "$schema": "http://json-schema.org/draft-07/schema#",
```

```
      "title": "astra.account.ldap",
      "type": "object",
      "properties": {
        "connectionHost": {
          "type": "string",
          "description": "The hostname or IP address of your LDAP server."
        },
        "credentialId": {
          "type": "string",
          "description": "The credential ID for LDAP account."
        },
        "groupBaseDN": {
          "type": "string",
          "description": "The base DN of the tree used to start the group
search. The system searches the subtree from the specified location."
        },
        "groupSearchCustomFilter": {
          "type": "string",
          "description": "Type of search that controls the default group
search filter used."
        },
        "isEnabled": {
          "type": "string",
          "description": "This property determines if this setting is
enabled or not."
        },
        "port": {
          "type": "integer",
          "description": "The port on which the LDAP server is running."
        },
        "secureMode": {
          "type": "string",
          "description": "The secure mode LDAPS or LDAP."
        },
        "userBaseDN": {
          "type": "string",
          "description": "The base DN of the tree used to start the user
search. The system searches the subtree from the specified location."
        },
        "userSearchFilter": {
          "type": "string",
          "description": "The filter used to search for users according a
search criteria."
        },
        "vendor": {
          "type": "string",
```

```
            "description": "The LDAP provider you are using.",
            "enum": ["Active Directory"]
          }
        },
        "additionalProperties": false,
        "required": [
          "connectionHost",
          "secureMode",
          "credentialId",
          "userBaseDN",
          "userSearchFilter",
          "groupBaseDN",
          "vendor",
          "isEnabled"
        ]
        },
        "state": "valid",
      }
    ],
    "metadata": {}
}
```

Locate the `state` field in the response which will have one of the values in the table below.

State	Description
pending	The configuration process is still active and not completed yet.
valid	Configuration has been completed successfully and `currentConfig` in the response matches `desiredConfig`.
error	The LDAP configuration process failed.

Add LDAP entries to Astra

After LDAP is configured as an authentication provider for Astra Control Center, you can select the LDAP users that Astra will authenticate using the LDAP credentials. Each user must have a role in Astra before they can access Astra through the Astra Control REST API.

There are two ways you can configure Astra to assign roles. Choose the one that is appropriate for your environment.

- Add and bind an individual user
- Add and bind a group

The LDAP credentials are in the form of a username as an email address and the associated LDAP password.

Add and bind an individual user

You can assign a role to each Astra user which is used after LDAP authentication. This is appropriate when there is a small number of users and each might have different administrative characteristics.

1. Add a user

Perform the following REST API call to add a user to Astra and indicate that LDAP is the authentication provider.

HTTP method	Path
POST	/account/{account_id}/core/v1/users

JSON input example

```
{
  "type" :  "application/astra-user",
  "version" : "1.1",
  "authID" : "cn=JohnDoe,ou=users,ou=astra,dc=example,dc=com",
  "authProvider" : "ldap",
  "firstName" : "John",
  "lastName" : "Doe",
  "email" : "john.doe@example.com"
}
```

Note the following about the input parameters:

- The following parameters are required:
  - `authProvider`
  - `authID`
  - `email`
- `authID` is the distinguished name (DN) of the user in LDAP
- `email` must be unique for all users defined in Astra

If the `email` value is not unique, an error occurs and a 409 HTTP status code is returned in the response.

Curl example

```
curl --location -i --request POST --data @JSONinput
'https://astra.example.com/accounts/<ACCOUNT_ID>/core/v1/users' --header
'Content-Type: application/astra-user+json' --header 'Accept: */*'
--header 'Authorization: Bearer <API_TOKEN>'
```

JSON response example


```
{
  "metadata": {
    "creationTimestamp": "2022-07-21T17:44:18Z",
    "modificationTimestamp": "2022-07-21T17:44:18Z",
    "createdBy": "8a02d2b8-a69d-4064-827f-36851b3e1e6e",
    "labels": []
  },
  "type": "application/astra-user",
  "version": "1.2",
  "id": "a7b5e674-a1b1-48f6-9729-6a571426d49f",
  "authProvider": "ldap",
  "authID": "cn=JohnDoe,ou=users,ou=astra,dc=example,dc=com",
  "firstName": "John",
  "lastName": "Doe",
  "companyName": "",
  "email": "john.doe@example.com",
  "postalAddress": {
    "addressCountry": "",
    "addressLocality": "",
    "addressRegion": "",
    "streetAddress1": "",
    "streetAddress2": "",
    "postalCode": ""
  },
  "state": "active",
  "sendWelcomeEmail": "false",
  "isEnabled": "true",
  "isInviteAccepted": "true",
  "enableTimestamp": "2022-07-21T17:44:18Z",
  "lastActTimestamp": ""
}
```


2. Add a role binding for the user

Perform the following REST API call to bind the user to a specific role. You need to have the UUID of the user created in the previous step.

HTTP method	Path
POST	/account/{account_id}/core/v1/roleBindings

JSON input example

```
{
  "type": "application/astra-roleBinding",
  "version": "1.1",
  "accountID": "{account_id}",
  "userID": "a7b5e674-a1b1-48f6-9729-6a571426d49f",
  "role": "member",
  "roleConstraints": ["*"]
}
```

Note the following about the input parameters:

- The value used above for `roleConstraint` is the only option available for the current release of Astra. It indicates the user is not restricted to a limited set of namespaces and can access them all.

Curl example

```
curl --location -i --request POST --data @JSONinput
'https://astra.example.com/accounts/<ACCOUNT_ID>/core/v1/roleBindings'
--header 'Content-Type: application/astra-roleBinding+json' --header
'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

JSON response example

```
{
  "metadata": {
    "creationTimestamp": "2022-07-21T18:08:24Z",
    "modificationTimestamp": "2022-07-21T18:08:24Z",
    "createdBy": "8a02d2b8-a69d-4064-827f-36851b3e1e6e",
    "labels": []
  },
  "type": "application/astra-roleBinding",
  "principalType": "user",
  "version": "1.1",
  "id": "b02c7e4d-d483-40d1-aaff-e1f900312114",
  "userID": "a7b5e674-a1b1-48f6-9729-6a571426d49f",
  "groupID": "00000000-0000-0000-0000-000000000000",
  "accountID": "d0fdbfa7-be32-4a71-b59d-13d95b42329a",
  "role": "member",
  "roleConstraints": ["*"]
}
```

Note the following about the response parameters:

- The value `user` for the `principalType` field indicates the role binding was added for a user (not a

group).

Add and bind a group

You can assign a role to an Astra group which is used after LDAP authentication. This is appropriate when there is a large number of users and each might have similar administrative characteristics.

1. Add a group

Perform the following REST API call to add a group to Astra and indicate that LDAP is the authentication provider.

HTTP method	Path
POST	/account/{account_id}/core/v1/groups

JSON input example

```
{
  "type": "application/astra-group",
  "version": "1.0",
  "name": "Engineering",
  "authProvider": "ldap",
  "authID": "CN=Engineering,OU=groups,OU=astra,DC=example,DC=com"
}
```

Note the following about the input parameters:

- The following parameters are required:
  - `authProvider`
  - `authID`

Curl example

```
curl --location -i --request POST --data @JSONinput
'https://astra.example.com/accounts/<ACCOUNT_ID>/core/v1/groups' --header
'Content-Type: application/astra-group+json' --header 'Accept: */*'
--header 'Authorization: Bearer <API_TOKEN>'
```

JSON response example

```
{
  "type": "application/astra-group",
  "version": "1.0",
  "id": "8b5b54da-ae53-497a-963d-1fc89990525b",
  "name": "Engineering",
  "authProvider": "ldap",
  "authID": "CN=Engineering,OU=groups,OU=astra,DC=example,DC=com",
  "metadata": {
    "creationTimestamp": "2022-07-21T18:42:52Z",
    "modificationTimestamp": "2022-07-21T18:42:52Z",
    "createdBy": "8a02d2b8-a69d-4064-827f-36851b3e1e6e",
    "labels": []
  }
}
```

2. Add a role binding for the group

Perform the following REST API call to bind the group to a specific role. You need to have the UUID of the group created in the previous step. Users that are members of the group will be able to sign in to Astra after LDAP performs the authentication.

HTTP method	Path
POST	/account/{account_id}/core/v1/roleBindings

JSON input example

```
{
  "type": "application/astra-roleBinding",
  "version": "1.1",
  "accountID": "{account_id}",
  "groupID": "8b5b54da-ae53-497a-963d-1fc89990525b",
  "role": "viewer",
  "roleConstraints": ["*"]
}
```

Note the following about the input parameters:

- The value used above for `roleConstraint` is the only option available for the current release of Astra. It indicates the user is not restricted to certain namespaces and can access them all.

Curl example

```
curl --location -i --request POST --data @JSONinput
'https://astra.example.com/accounts/<ACCOUNT_ID>/core/v1/roleBindings'
--header 'Content-Type: application/astra-roleBinding+json' --header
'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

JSON response example


```
{
  "metadata": {
    "creationTimestamp": "2022-07-21T18:59:43Z",
    "modificationTimestamp": "2022-07-21T18:59:43Z",
    "createdBy": "527329f2-662c-41c0-ada9-2f428f14c137",
    "labels": []
  },
  "type": "application/astra-roleBinding",
  "principalType": "group",
  "version": "1.1",
  "id": "2f91b06d-315e-41d8-ae18-7df7c08fbb77",
  "userID": "00000000-0000-0000-0000-000000000000",
  "groupID": "8b5b54da-ae53-497a-963d-1fc89990525b",
  "accountID": "d0fdbfa7-be32-4a71-b59d-13d95b42329a",
  "role": "viewer",
  "roleConstraints": ["*"]
}
```


Note the following about the response parameters:

- The value `group` for the `principalType` field indicates the role binding was added for a group (not a user).

Disable and reset LDAP

There are two optional though related administrative tasks you can perform as needed for an Astra Control Center deployment. You can globally disable LDAP authentication and reset the LDAP configuration.

Both workflow tasks require the id for the `astra.account.ldap` Astra setting. Details for how to retrieve the setting id are included in **Configure the LDAP server**. See Retrieve the UUID of the LDAP setting for more information.

- Disable LDAP authentication
- Reset the LDAP authentication configuration

Disable LDAP authentication

You can perform the following REST API call to globally disable LDAP authentication for a specific Astra deployment. The call updates the `astra.account.ldap` setting and the `isEnabled` value is set to `false`.

HTTP method	Path
PUT	/account/{account_id}/core/v1/settings/{setting_id}

JSON input example

```
{
  "type": "application/astra-setting",
  "version": "1.0",
  "desiredConfig": {
    "connectionHost": "myldap.example.com",
    "credentialId": "3bd9c8a7-f5a4-4c44-b778-90a85fc7d154",
    "groupBaseDN": "OU=groups,OU=astra,DC=example,DC=com",
    "isEnabled": "false",
    "port": 686,
    "secureMode": "LDAPS",
    "userBaseDN": "OU=users,OU=astra,DC=example,dc=com",
    "userSearchFilter": "((objectClass=User))",
    "vendor": "Active Directory"
    }
}
```

```
curl --location -i --request PUT --data @JSONinput
'https://astra.example.com/accounts/<ACCOUNT_ID>/core/v1/settings/<SETTING
_ID>' --header 'Content-Type: application/astra-setting+json' --header
'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

If the call is successful, the `HTTP 204` response is returned. You can optionally retrieve the configuration settings again to confirm the change.

Reset the LDAP authentication configuration

You can perform the following REST API call to disconnect Astra from the LDAP server and reset the LDAP configuration in Astra. The call updates the `astra.account.ldap` setting and the value of `connectionHost` is cleared.

The value of `isEnabled` must also be set to `false`. You can either set this value before making the reset call or as part of making the reset call. In the second case, `connectionHost` should be cleared and `isEnabled` set to false on the same reset call.

This is a disruptive operation and you should proceed with caution. It deletes all the imported LDAP users and groups. It also deletes all the related Astra users, groups, and roleBindings (LDAP type) that you created in Astra Control Center.

HTTP method	Path
PUT	/account/{account_id}/core/v1/settings/{setting_id}

JSON input example

```
{
  "type": "application/astra-setting",
  "version": "1.0",
  "desiredConfig": {
    "connectionHost": "",
    "credentialId": "3bd9c8a7-f5a4-4c44-b778-90a85fc7d154",
    "groupBaseDN": "OU=groups,OU=astra,DC=example,DC=com",
    "isEnabled": "false",
    "port": 686,
    "secureMode": "LDAPS",
    "userBaseDN": "OU=users,OU=astra,DC=example,dc=com",
    "userSearchFilter": "((objectClass=User))",
    "vendor": "Active Directory"
    }
}
```

Note the following:

- To change the LDAP server, you must both disable and reset LDAP changing `connectHost` to a null value as shown in the example above.

```
curl --location -i --request PUT --data @JSONinput
'https://astra.example.com/accounts/<ACCOUNT_ID>/core/v1/settings/<SETTING
_ID>' --header 'Content-Type: application/astra-setting+json' --header
'Accept: */*' --header 'Authorization: Bearer <API_TOKEN>'
```

If the call is successful, the `HTTP 204` response is returned. You can optionally retrieve the configuration again to confirm the change.

Buckets

List buckets

You can list the S3 buckets defined for a specific Astra account.

1. List the buckets

Perform the following REST API call.

HTTP method	Path
GET	/account/{account_id}/topology/v1/buckets

Curl example: Return all data for all buckets

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

Storage

List storage backends

You can list the available storage backends.

1. List the backends

Perform the following REST API call.

HTTP method	Path
GET	/account/{account_id}/topology/v1/storageBackends

Curl example: Return all data for all storage backends

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

JSON output example

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


Clusters

List managed clusters

You can list the Kubernetes clusters currently managed by Astra.

1. List the clusters

Perform the following REST API call.

HTTP method	Path
GET	/account/{account_id}/topology/v1/managedClusters

Curl example: Return all data for all clusters

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

Management workflows

Before you begin

You can use these workflows as part of administering the applications within an Astra managed cluster.

These workflows can be expanded and enhanced by NetApp at any time and so you should review them periodically.

General preparation

Before using any of the Astra workflows, make sure to review Prepare to use the workflows.

Workflow categories

The management workflows are organized in different categories to make it easier to locate the one you want.

Category	Description
Application control	These workflows allow you to control the managed and unmanaged applications. You can list the apps as well as create and remove a managed app.
Application protection	You can use these workflows to protect your managed applications through snapshots and backups.
Cloning and restoring apps	These workflow describe how to clone and restore your managed applications.
Support	There are several workflows available to debug and support your applications as well as the general Kubernetes environment.

Additional considerations

There are a several additional considerations when using the management workflows.

Cloning an app

There are a few things to consider when cloning an application. The parameters described below are part of the JSON input.

Source cluster identifier

The value of `sourceClusterID` always identifies the cluster where the original app is installed.

Cluster identifier

The value of `clusterID` identifies the cluster where the new app will be installed.

- When cloning within the same cluster, `clusterID` and `sourceClusterID` have the same value.
- When cloning across clusters, the two values are different and `clusterID` should be the ID of the target cluster.

Namespaces

The `namespace` value must be different than the original source app. Further, the namespace for the clone cannot exist and Astra will create it.

Backups and snapshots

You can optionally clone an application from an existing backup or snapshot using the `backupID` or `snapshotID` parameters. If you don't provide a backup or snapshot, Astra will create a backup of the application first and then clone from the backup.

Restoring an app

Here are a few things to consider when restoring an application.

- Restoring an application is very similar to the clone operation.
- When restoring an app, you must provide either a backup or snapshot.

App control

List the unmanaged apps

You can list the applications that are currently not managed by Astra. You might do this as part of selecting an app to be managed.

The REST endpoint used in these workflows returns all the Astra applications by default. You can use the `filter` query parameter on the API call to request only the unmanaged apps be returned. As an alternative, you can omit the filter parameter to return all the apps and then examine the `managedState` field in the output to determine which apps are in the `unmanaged` state.

List only the apps with managedState equal to unmanaged

This workflow uses the `filter` query parameter to return only the unmanaged apps.

1. List the unmanaged applications

Perform the following REST API call.

HTTP method	Path
GET	/account/{account_id}/topology/v1/apps

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
filter	Query	No	Use a filter to specify which apps should be returned.

Parameter	Type	Required	Description
include	Query	No	Optionally select the values you want returned in the response.

Curl example: Return the name, id, and managedState for the unmanaged apps

```
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 output example

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


List all the apps and select the unmanaged apps

This workflow returns all the apps. You must examine the output to determine which are unmanaged.

1. List all the applications

Perform the following REST API call.

HTTP method	Path
GET	/account/{account_id}/topology/v1/apps

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
include	Query	No	Optionally select the values you want returned in the response.

Curl example: Return all data for all apps

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

Curl example: Return the name, id, and managedState for all apps

```
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 output example

```
{
  "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. Select the unmanaged applications

Review the output of the API call and manually select the apps with `managedState` equal to `unmanaged`.

List the managed apps

You can list the applications that are currently managed by Astra. You might do this as part of finding the snapshots or backups for a specific app.

1. List the applications

Perform the following REST API call.

HTTP method	Path
GET	/account/{account_id}/k8s/v1/managedApps

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
include	Query	No	Optionally select the values you want returned in the response.

Curl example: Return all data for all apps

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

Curl example: Return the name, id, and state for all apps

```
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 output example

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


Get a managed app

You can retrieve all the resource variables describing a single managed application.

Before you begin

You must have the ID of the managed app you want to retrieve. If needed you can use the workflow List the managed apps to locate the application.

1. Get the application

Perform the following REST API call.

HTTP method	Path
GET	/accounts/{account_id}/k8s/v1/managedApps/{managedApp_id}

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
managed app id	Path	Yes	ID value of the managed application to retrieve.

Curl example: Return all data for the application

```
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>'
```

Manage an app

You can create a managed application based on an application already known to Astra.

When an application is managed, you can protect it by taking regular backups and snapshots.

Before you begin

You must have the ID of the discovered app you want to manage. If needed you can use the workflow List the unmanaged apps to locate the application.

1. Manage the application

Perform the following REST API call.

HTTP method	Path
POST	/account/{account_id}/k8s/v1/managedApps

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
JSON	Body	Yes	Provides the parameters needed to identify the application to be managed. See the example below.

JSON input example

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

Curl example: Manage an app

```
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
```

Unmanage an app

You can remove a managed app when it's no longer needed. Removing a managed application also deletes the associated schedules.

Before you begin

You must have the ID of the managed app you want to unmanage. If needed you can use the workflow List the managed apps to locate the application.

The application's backups and snapshots are not automatically removed when it is deleted. If you no longer need the backups and snapshots, you should delete them before removing the application.

1. Unmanaged the app

Perform the following REST API call.

HTTP method	Path
DELETE	/accounts/{account_id}/k8s/v1/managedApps/{managedApp_id}

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
managed app id	Path	Yes	Identifies the managed application to remove.

Curl example: Remove a managed app

```
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>'
```

App protection

List the snapshots

You can list the snapshots that have been taken for a specific managed application.

Before you begin

You must have the ID of the managed app you want to list the snapshots for. If needed you can use the workflow List the managed apps to locate the application.

1. List the snapshots

Perform the following REST API call.

HTTP method	Path
GET	/accounts/{account_id}/k8s/v1/managedApps/{managedApp_id}/appSnaps

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
managed app id	Path	Yes	Identifies the managed application owning the listed snapshots.
count	Query	No	If `count=true` the number of snapshots is included in the metadata section of the response.

Curl example: Return all snapshots for the app

```
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 example: Return all snapshots for the app and the count

```
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 output example

```
{
  "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
  }
}
```


List the backups

You can list the backups that have been created for a specific managed application.

Before you begin

You must have the ID of the managed app you want to list the backups for. If needed you can use the workflow List the managed apps to locate the application.

1. List the backups

Perform the following REST API call.

HTTP method	Path
GET	/accounts/{account_id}/k8s/v1/managedApps/{managedApp_id}/appBackups

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
managed app id	Path	Yes	Identifies the managed application owning the listed backups.

Curl example: Return all backups for the app

```
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 output example


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


Create a snapshot for a managed app

You can create a snapshot for a specific managed application.

Before you begin

You must have the ID of the managed app you want to create a snapshot for. If needed you can use the workflow List the managed apps to locate the application.

1. Create a snapshot

Perform the following REST API call.

HTTP method	Path
POST	/accounts/{account_id}/k8s/v1/managedApps/{managedApp_id}/appSnaps

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
managed app id	Path	Yes	Identifies the managed application where the snapshot will be created.
JSON	Body	Yes	Provides the parameters for the snapshot. See the example below.

JSON input example

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

Curl example: Create a snapshot for the app

```
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
```

Create a backup for a managed app

You can create a backup for a specific managed application. You can use the backup to restore or clone the app.

Before you begin

You must have the ID of the managed app you want to create a backup for. If needed you can use the workflow List the managed apps to locate the application.

1. Create a backup

Perform the following REST API call.

HTTP method	Path
POST	/accounts/{account_id}/k8s/v1/managedApps/{managedApp_id}/appBackups

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
managed app id	Path	Yes	Identifies the managed application where the backup will be created.
JSON	Body	Yes	Provides the parameters for the backup. See the example below.

JSON input example

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

Curl example: Create a backup for the app

```
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
```

Delete a snapshot

You can delete a snapshot associated with a managed application.

Before you begin

You must have the following:

- ID of the managed app that owns the snapshot. If needed you can use the workflow List the managed apps to locate the application.
- ID of the snapshot you want to delete. If needed you can use the workflow List the snapshots to locate the snapshot.

1. Delete the snapshot

Perform the following REST API call.

HTTP method	Path
DELETE	/accounts/{account_id}/k8s/v1/managedApps/{managedApp_id}/appSnaps/{appSnap_id}

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
managed app id	Path	Yes	Identifies the managed application owning the snapshot.
snapshot id	Path	Yes	Identifies the snapshot to be deleted.

Curl example: Delete a single snapshot for the app

```
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>'
```

Delete a backup

You can delete a backup associated with a managed application.

Before you begin

You must have the following:

- ID of the managed app that owns the backup. If needed you can use the workflow List the managed apps to locate the application.
- ID of the backup you want to delete. If needed you can use the workflow List the backups to locate the snapshot.

1. Delete the backup

Perform the following REST API call.

> You can force the deletion of a failed backup using the optional request header as described below.

HTTP method	Path
DELETE	/accounts/{account_id}/k8s/v1/managedApps/{managedApp_id}/appBackups/{appBackup_id}

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
managed app id	Path	Yes	Identifies the managed application owning the backup.
backup id	Path	Yes	Identifies the backup to be deleted.
force delete	Header	No	Used to force the deletion of a failed backup.

Curl example: Delete a single backup for the app

```
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 example: Delete a single backup for the app with the force option

```
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'
```

Cloning and restoring an app

Clone a managed app

You can create a new application by cloning an existing managed app.

Before you begin

Note the following about this workflow:

- An app backup or snapshot is not used
- The clone operation is performed within the same cluster

> To clone an app to a different cluster, you need to update the `clusterId` parameter in the JSON input as appropriate for your environment.

1. Select the managed app to clone

Perform the workflow List the managed apps and select application you want to clone. Several of the resource values are needed for the REST call used to clone the app.

2. Clone the app

Perform the following REST API call.

HTTP method	Path
POST	/account/{account_id}/k8s/v1/managedApps

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
JSON	Body	Yes	Provides the parameters for the cloned app. See the example below.

JSON input example

```
{
  "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 example: Clone an app

```
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
```

Clone a managed app from a snapshot

You can create a new application by cloning it from an app snapshot.

Before you begin

Note the following about this workflow:

- An app snapshot is used
- The clone operation is performed within the same cluster

To clone an app to a different cluster, you need to update the `clusterId` parameter in the JSON input as appropriate for your environment.

1. Select the managed app to clone

Perform the workflow List the managed apps and select application you want to clone. Several of the resource values are needed for the REST call used to clone the app.

2. Select the snapshot to use

Perform the workflow List the snapshots and select snapshot you want to use.

3. Clone the app

Perform the following REST API call.

HTTP method	Path
POST	/account/{account_id}/k8s/v1/managedApps

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
JSON	Body	Yes	Provides the parameters for the cloned app. See the example below.

JSON input example

```
{
  "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 example: Clone an app from a snapshot

```
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
```

Clone a managed app from a backup

You can create a new managed application by cloning it from an app backup.

Before you begin

Note the following about this workflow:

- An app backup is used
- The clone operation is performed within the same cluster

> To clone an app to a different cluster, you need to update the `clusterId` parameter in the JSON input as appropriate for your environment.

1. Select the managed app to clone

Perform the workflow List the managed apps and select application you want to clone. Several of the resource values are needed for the REST call used to clone the app.

2. Select the backup to use

Perform the workflow List the backups and select backup you want to use.

3. Clone the app

Perform the following REST API call.

HTTP method	Path
POST	/account/{account_id}/k8s/v1/managedApps

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
JSON	Body	Yes	Provides the parameters for the cloned app. See the example below.

JSON input example

```
{
  "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 example: Clone an app from a backup

```
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
```

Restore a managed app from a backup

You can restore a managed application by creating a new app from a backup.

1. Select the managed app to restore

Perform the workflow List the managed apps and select application you want to clone. Several of the resource values are needed for the REST call used to clone the app.

2. Select the backup to use

Perform the workflow List the backups and select backup you want to use.

3. Restore the app

Perform the following REST API call. You must provide the ID for either a backup (as shown below) or snapshot.

HTTP method	Path
PUT	/account/{account_id}/k8s/v1/managedApps/{app_id}

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
JSON	Body	Yes	Provides the parameters for the cloned app. See the example below.

JSON input example

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

Curl example: Restore an app in place from a backup

```
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
```

Support

List the notifications

You can list the notifications for a specific Astra account. You might do this as part of monitoring the system activity or debugging an issue.

1. List the notifications

Perform the following REST API call.

HTTP method	Path
GET	/account/{account_id}/core/v1/notifications

Additional input parameters

In addition to the parameters common with all REST API calls, the following parameters are also used in the curl examples for this step.

Parameter	Type	Required	Description
filter	Query	No	Optionally filter the notifications you want returned in the response.
include	Query	No	Optionally select the values you want returned in the response.

Curl example: Return all notifications

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

Curl example: Return the description for notifications with severity of warning

```
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 output example

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

Delete a failed app

You might be unable to remove a managed app if it has a backup or snapshot in a failed state. In this case you can manually remove the app using the workflow described below.

1. Select the managed app to delete

Perform the workflow List the managed apps and select application you want to remove.

2. List the existing backups for the app

Perform the workflow List the backups.

3. Delete all the backups

Delete all the app backups by performing the workflow Delete a backup for each backup in the list.

4. List the existing snapshots for the app

Perform the workflow List the snapshots.

5. Delete all the snapshots

Perform the workflow Delete a snapshot from each snapshot in the list.

6. Remove the application

Perform the workflow Unmanage an app to remove the application.

Using Python

NetApp Astra Control Python SDK

NetApp Astra Control Python SDK is an open source package you can use to automate an Astra Control deployment. The package is also a valuable resource for learning about the Astra Control REST API, perhaps as part of creating your own automation platform.

For simplicity, the NetApp Astra Control Python SDK will be referred to as the **SDK** throughout the remainder this page.

Two related software tools

The SDK includes two different though related tools which operate at different levels of abstraction when accessing the Astra Control REST API.

Astra SDK

The Astra SDK provides the core platform functionality. It includes a set of Python classes which abstract the underlying REST API calls. The classes support administrative actions on various Astra Control resources, including apps, backups, snapshots, and clusters.

The Astra SDK is one part of the package and is provided in the single `astraSDK.py` file. You can import this file into your environment and use the classes directly.

The **NetApp Astra Control Python SDK** (or just SDK) is the name of the entire package. The **Astra SDK** refers to the core Python classes in the single file `astraSDK.py`.

Toolkit script

In addition to the Astra SDK file, the `toolkit.py` script is also available. This script operates at a higher level of abstraction by providing access to discrete administrative actions defined internally as Python functions. The script imports the Astra SDK and makes calls to the classes as needed.

How to access

You can access the SDK in the following ways.

Python package

The SDK is available at Python Package Index under the name **netapp-astra-toolkits**. The package is assigned a version number and will continue to be updated as needed. You must use the **PiP** package management utility to install the package into your environment.

See PyPI: NetApp Astra Control Python SDK for more information.

GitHub source code

The SDK source code is also available at GitHub. The repository includes the following:

- `astraSDK.py` (Astra SDK with Python classes)
- `toolkit.py` (higher level function-based script)

- Detailed installation requirements and instructions
- Installation scripts
- Additional documentation

You can clone the GitHub: Netapp/netapp-astra-toolkits repository to your local environment.

Installation and basic requirements

There are several options and requirements to consider as part of installing the package and preparing to use it.

Summary of the installation options

You can install the SDK in one of the following ways:

- Use Pip to install the package from PyPI into your Python environment
- Clone the Git Hub repository and either:
  - Deploy the package as a Docker container (which includes everything you need)
  - Copy the two core Python files so they are accessible to your Python client code

Refer to the PyPI and GitHub pages for more information.

Requirements for the Astra Control environment

Whether directly using the Python classes in the Astra SDK or the functions in the `toolkit.py` script, ultimately you'll be accessing the REST API at an Astra Control deployment. Because of this you'll need an Astra account along with an API token. See Before you begin and the other pages in the **Get started** section of this documentation for more information.

Requirements for the NetApp Astra Control Python SDK

The SDK has several prerequisites related to the local Python environment. For example, you must use Python 3.5 or later. In addition, there are several Python packages that are required. See the GitHub repository page or PyPI package page for more information.

Summary of helpful resources

Here are some the resources you'll need to get started.

- PyPI: NetApp Astra Control Python SDK
- GitHub: Netapp/netapp-astra-toolkits

Native Python

Before you begin

Python is a popular development language especially for datacenter automation. Before using the native features of Python together with several common packages, you need to prepare the environment and the required input files.

In addition to accessing the Astra Control REST API directly using Python, NetApp also provides a toolkit package which abstracts the API and removes some of the complexities. See NetApp Astra Control Python SDK for more information.

Prepare the environment

The basic configuration requirements to run the Python scripts are described below.

Python 3

You need to have the latest version of Python 3 installed.

Additional libraries

The **Requests** and **urllib3** libraries must be installed. You can use pip or another Python management tool as appropriate for your environment.

Network access

The workstation where the scripts run must have network access and be able to reach Astra Control. When using Astra Control Service, you must be connected to the internet and be able to connect to the service at `https://astra.netapp.io`.

Identity information

You need a valid Astra account with the account identifier and API token. See Get an API token for more information.

Create the JSON input files

The Python scripts rely on configuration information contained in JSON input files. Sample files are provided below.

You need to update the samples as appropriate for your environment.

Identity information

The following file contains the API token and Astra account. You need to pass this file to Python scripts using the `-i` (or `--identity`) CLI parameter.

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

List the managed apps

You can use the following script to list the managed applications for your Astra account.

See Before you begin for an example of the required JSON input file.

```
1 #!/usr/bin/env python3
```

```
2
```

```
##-------------------------------------------------------------------------
------
  3 #
  4 # Usage: python3 list_man_apps.py -i identity_file.json
  5 #
  6 # (C) Copyright 2021 NetApp, Inc.
  7 #
  8 # This sample code is provided AS IS, with no support or warranties of
  9 # any kind, including but not limited for warranties of
merchantability
 10 # or fitness of any kind, expressed or implied. Permission to use,
 11 # reproduce, modify and create derivatives of the sample code is
granted
 12 # solely for the purpose of researching, designing, developing and
 13 # testing a software application product for use with NetApp products,
 14 # provided that the above copyright notice appears in all copies and
 15 # that the software application product is distributed pursuant to
terms
 16 # no less restrictive than those set forth herein.
 17 #
 18
##-------------------------------------------------------------------------
------
 19
 20 import argparse
 21 import json
 22 import requests
 23 import urllib3
 24 import sys
 25
 26 # Global variables
 27 api_token = ""
 28 account_id = ""
 29
 30 def get_managed_apps():
 31     ''' Get and print the list of managed apps '''
 32
 33     # Global variables
 34     global api_token
 35     global account_id
 36
 37     # Create an HTTP session
 38     sess1 = requests.Session()
 39
 40     # 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
```

```
84      sess1.close()
85
86      return
87
88  def read_id_file(idf):
89      ''' Read the identity file and save values '''
90
91      # Global variables
92      global api_token
93      global account_id
94
95      with open(idf) as f:
96          data = json.load(f)
97
98      api_token = data['api_token']
99      account_id = data['account_id']
100
101     return
102
103 def main(args):
104     ''' Main top level function '''
105
106     # Global variables
107     global api_token
108     global account_id
109
110     # Retrieve name of JSON input file
111     identity_file = args.id_file
112
113     # Get token and account
114     read_id_file(identity_file)
115
116     # Issue REST call
117     get_managed_apps()
118
119     return
120
121 def parseArgs():
122         ''' Parse the CLI input parameters '''
123
124         parser = argparse.ArgumentParser(description='Astra REST API -
    List the managed apps',
125                                 add_help = True)
126         parser.add_argument("-i", "--identity", action="store", dest
    ="id_file", default=None,
127                                 help='(Req) Name of the identity input
```

```
    file', required=True)
128
129         return parser.parse_args()
130
131 if __name__ == '__main__':
132     ''' Begin here '''
133
134     # Parse input parameters
135     args = parseArgs()
136
137     # Call main function
138     main(args)
```

API reference

You can access the details of all the Astra Control REST API calls, including the HTTP methods, input parameters, and responses. This complete reference is helpful when developing automation applications using the REST API.

The REST API reference documentation is currently provided with Astra Control and is available online.

Before you begin

You need an account for Astra Control Center or Astra Control Service.

Steps

1. Sign in to Astra using your account credentials.

   Access the following site for Astra Control Service: https://astra.netapp.io

2. Click the figure icon at the top right of the page and select **API access**.
3. At the top of the page click the URL displayed under **API Documentation**.
4. Provide your account credentials again if prompted.

Additional resources

There are additional resources you can access to get help and find more information about NetApp cloud services and support as well as general REST and cloud concepts.

Astra

- Astra Control Center documentation

  Documentation for the current release of the Astra Control Center software deployed on the customer premises.

- Astra Control Service documentation

  Documentation for the current release of the Astra Control Service software available in the public cloud.

- Astra Data Store documentation

  Documentation for the current release of the Astra Data Store software deployed on the customer premises.

- Astra Trident documentation

  Documentation for the current release of the Astra Trident software, an open source storage orchestrator maintained by NetApp.

- Astra family documentation

  Central location for accessing all the Astra documentation for both on-premises and public cloud deployments.

NetApp cloud resources

- NetApp Cloud Solutions

  Central site for the NetApp cloud solutions.

- NetApp Cloud Central console

  NetApp Cloud Central service console with sign in.

- NetApp Support

  Access troubleshooting tools, documentation, and technical support assistance.

REST and cloud concepts

- PhD dissertation by Roy Fielding

  This publication introduced and established the REST application development model.

- Auth0

  This is the authentication and authorization platform service used by the Astra service for web access.

- RFC editor

  Authoritative source for web and Internet standards maintained as a collection of uniquely numbered RFC documents.

Earlier versions of Astra Control Automation documentation

You can access the automation documentation for previous Astra Control releases at the links below.

- Astra Control Automation 22.04 documentation
- Astra Control Automation 21.12 documentation
- Astra Control Automation 21.08 documentation

Legal notices

Legal notices provide access to copyright statements, trademarks, patents, and more.

Copyright

http://www.netapp.com/us/legal/copyright.aspx

Trademarks

NETAPP, the NETAPP logo, and the marks listed on the NetApp Trademarks page are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.

http://www.netapp.com/us/legal/netapptmlist.aspx

Patents

A current list of NetApp owned patents can be found at:

https://www.netapp.com/us/media/patents-page.pdf

Privacy policy

https://www.netapp.com/us/legal/privacypolicy/index.aspx

Astra Control API license

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

Copyright Information

Copyright © 2022 NetApp, Inc. All rights reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means-graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system-without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7103 (October 1988) and FAR 52-227-19 (June 1987).

Trademark Information

NETAPP, the NETAPP logo, and the marks listed at http://www.netapp.com/TM are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.