Astra automation documentation

Astra Automation

NetApp
June 06, 2021

Table of Contents

Astra automation documentation

Introduction to the Astra REST API

The NetApp Astra service provides a 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 on each API request.

Built on REST technology

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

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 can be used to access the resources collections. Some of the supported operations include:

- Filtering
- Sorting
- Pagination

Alignment with the Astra web UI

The Astra web interface uses the same REST API and so there is consistency between the two access paths and user experience.

Robust debugging and problem determination data

The Astra REST API provides a robust debugging and problem determination capability, including events and notifications generated internally.

Foundation for advanced automation technologies

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

Get started

You can quickly get started using the Astra REST API by first reviewing the basic preparation requirements in *Before you begin*. After that you'll need to generate an API token at the Astra web interface and use it with the basic Curl example in *Hello world*.

Before you begin

There are several steps you should take to prepare to use the Astra REST API.

Review REST concepts and implementation

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

Have Cloud Central account credentials

You'll need the **Auth0** account credentials to sign in to the Astra web user interface and generate an API token.

Get more information

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

Get an API access token

You must have an API access token to access the Astra REST API.

The Astra API tokens never expire, however you can revoke a token when it is no longer needed.

Before you begin

You need an account for the Astra service.

About this task

This task generates an API token at the Astra web interface. You should also retrieve the account ID.

Steps

1. Sign in to the Astra service using your account credentials.

   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.

After you finish

When you access the Astra REST API through Curl or a programming language, you must include in the API

bearer token in the `Authorization` request header. To revoke the token, see Security.

Hello world

You can issue a simple Curl command at your workstation's CLI to get started using the Astra 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 access 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>'
```

Core REST implementation

REST establishes a common set of technologies and best practices, however the details of each API can vary based on the choices made during development. You should be aware of the design characteristics of the Astra REST API before using it with a live cloud deployment.

Also see Additional considerations for more details on using the Astra REST API.

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 technologies and best practices for exposing server-based resources and managing their states. It uses mainstream protocols and standards to provide a flexible foundation for administering Astra deployments.

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 REST API uses JSON to format the data carried in the body of each HTTP request and response.

Resources and collections

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

API transactions and the CRUD model

The Astra 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 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.
HEAD	Read	Essentially issues a GET request but only returns the HTTP response headers.

Request and response headers

The following table summaries the HTTP headers used with the Astra REST API. See RFC 7232 for more information.

Header	Type	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.

Header	Type	Notes
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.
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 REST API are described below.

The Astra 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.

Code	Meaning	Description
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.

Format of the URL

The format of the URL with an example is presented below.

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`.

Additional considerations when using the REST API

There are several additional characteristics of the Astra REST API affecting its operation and use. You should be aware of these considerations before issuing an API call.

Security

The Astra REST API provides multiple layers of security.

All HTTP network traffic is protected using the transport layer security (TLS) standard.

Astra API tokens

To use the Astra REST API, you must provide an API token on every request in the `Authorization` request header. Note the following:

- You can generate an API token at the Astra web user interface.
- A token never expires after it is created.
- You can revoke a token at any time at the Astra web user interface.

See Get an API access token for more information.

Revoking an API access token

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

Before you begin

You need an account for the Astra service. 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 the Astra service using your account credentials.

   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**.

Roles and access control

Each Astra user is assigned to a single role. The role is used to restrict access to the API calls. The defined roles include:

- Owner

- Admin
- Member
- Viewer

Working with collections

The Astra 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 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.

Resources

You can use the resources provided through REST API to automate the administration of an Astra deployment. Each resource is access through one or more endpoints. This section provides an introduction to these resources which is useful as part of planning an automation deployment.

See API reference for more information.

Summary of resources for the Astra REST API

The resource endpoints provided in the Astra REST API are organized in three categories

Core resources

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

- Credential
- Event
- Notification
- Role Binding
- Token
- Unread notification
- User

Managed application resources

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

- Application asset
- Application backup
- Application snapshot
- Managed application
- Schedule

Topology resources

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

- App
- Volume

Format of the access path and URL

The format of the path and full URL used to access the Astra resources is based on several values. See URL format for more information.

API reference

You can access the details of all the Astra 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 available online with the Astra service.

Before you begin

You need an account for the Astra service.

Steps

1. Sign in to the Astra service using your account credentials.

   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.

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 concepts and cloud technology

- 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.

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 API license

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

Open source

Notice files provide information about third-party copyright and licenses used in NetApp software.

Notice for Astra