



Core REST implementation

Astra Automation

NetApp
June 06, 2021

Table of Contents

- Core REST implementation 1
 - REST web services 1
 - Resources and collections 2
 - HTTP details 3
 - URL format 6

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 <code>Location</code> 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 summarizes 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, <code>application/json</code> 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 <code>Accept</code> 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 authenticate.

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.

Copyright Information

Copyright © 2021 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.