Hewlett Packard Enterprise

TSS Paris 2019

HPE Composable Infrastructure Hands-On Lab

Programing HPE OneView and HPE Global Dashboard

Contents

Programing HPE OneView and HPE Global Dashboard Hands-On Lab

Summary

In December 2015, in London, HPE announced Synergy, the first platform architected from the ground up for composability. Earlier that year, HPE had announced the Composable API, and the Composable ecosystem.
This ecosystem is now growing because more and more software partners have expressed interest in integrating with the Composable API. Some have already demonstrated products, some have products in development, but all of them are using the same Composable API, also referred to as the HPE OneView API.
This API allows HPE OneView and the Synergy Composer (powered by HPE OneView) to be controlled by another software entity, thus it is an essential step in building a Software-Defined infrastructure. The controlling software entity might be as simple as a small script (PowerShell, Python, etc.) it might also be a configuration management system, such as Chef, Ansible or any kind of software that needs to interact with the underlying infrastructure (VMware vCenter, Microsoft System Center, Docker Machine, etc.)
In all cases an API is a prerequisite for a Software-Defined infrastructure and the growing ecosystem around it. HPE OneView and the HPE Synergy Composer is no exception and it comes with an API since v1.0 in September 2013.

REST API

REST (Representational State Transfer) is a style of API that uses basic Create, Read, Update and Delete (CRUD) operations that are performed on resources using HTTP POST, GET, PUT and DELETE requests. To learn more about general REST concepts, see:

http://en.wikipedia.org/wiki/Representational_state_transfer

HPE OneView has a resource-oriented architecture that provides a uniform REST interface. Every resource has one Uniform Resource Identifier (URI) and represents a physical device or logical construct, and may be manipulated using REST APIs.

Resource operations

Basic Create, Read, Update and Delete (CRUD) operations are performed on the appliance resources via the standard HTTP POST, GET, PUT and DELETE methods. RESTful interfaces are based on the world wide web standards, thus most modern web servers can support these operations without modification. Restful APIs are stateless. The resource state is maintained by the resource manager and is reported as the resource representation. Any application state must be maintained by the client and it may manipulate the resource locally, but until a PUT or POST is made, the resource as known by the resource manager is not changed.

Table 1. REST HTTP Operations

Operation	HTTP Verb	Description
Create	POST URI <Payload = Resource data>	New resources are created using the POST operation and including relevant data in the payload. On Success the Resource URI is returned.
Read	GET URI	Returns the requested resource representation(s)
Update	PUT URI <Payload = Update data>	Update an existing resource using the update data.
Delete	DELETE URI	Delete the addressed resource

URI format

All the appliance URIs point to resources and the client does not need to modify or create URIs. The URI for a specific resource is static and follows this format: `https://{appl}/rest/{resource name}`. The three parts are described below.

Table 2. URI Format

https://{appl} .	The appliance address
/rest	Type of URI.
/{resource name]	Name of the appliance resource such as server-profiles.

REST API calls can be executed using many different scripting languages such as PowerShell, Python, Java, Ruby, Golang and many other scripting and/or programming languages. Other tools like Postman or HTTPRequester for Firefox can be also very useful as they can place REST calls without writing any code, useful when you need to verify a REST call or quickly explore the JSON information you receive in return.

Data transfer format

The appliance resources support JSON (JavaScript Object Notation) as the standard for exchanging data using a REST API. If JSON is not specified in the REST API call, then the default is JSON.

To learn more about JSON, go to www.json.org

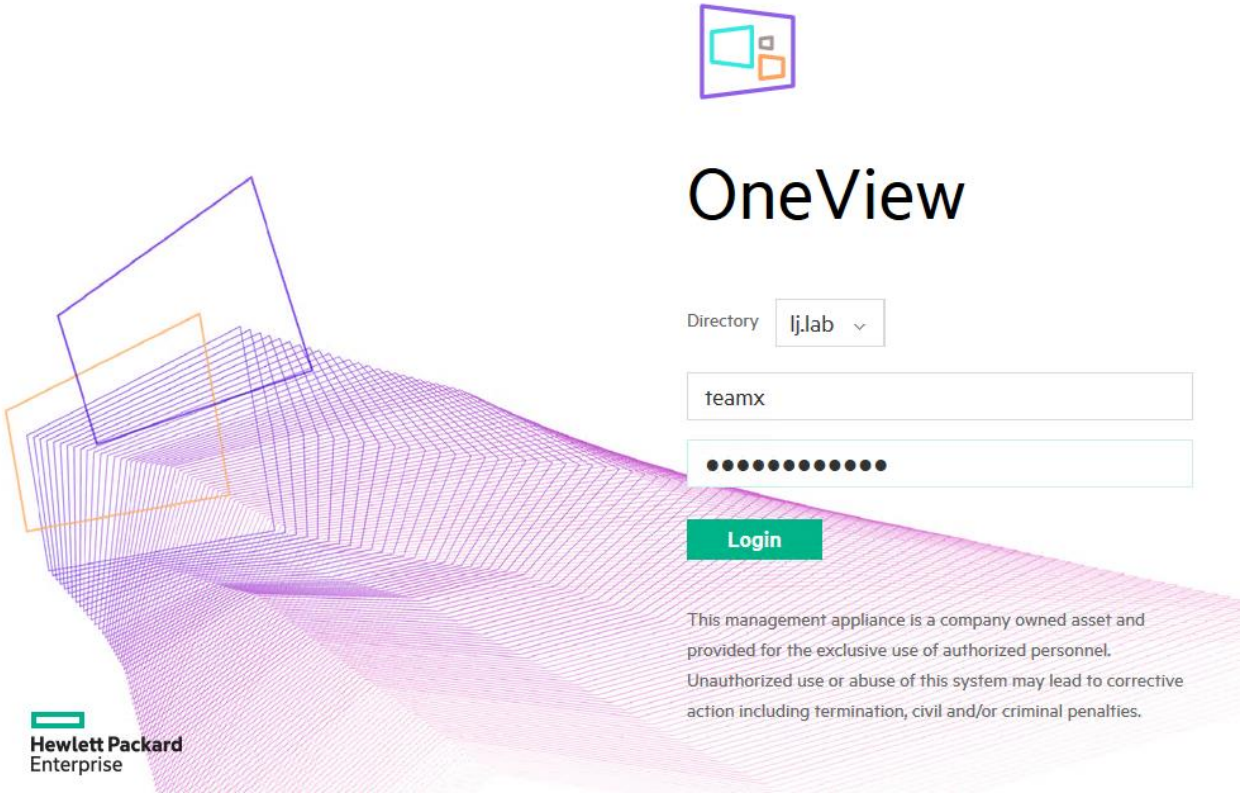
Lab 1: Read the Fantastic Manual

To make our first call to the HPE Composable API, we can refer to the fantastic manual provided with HPE OneView. This is where everyone should start!

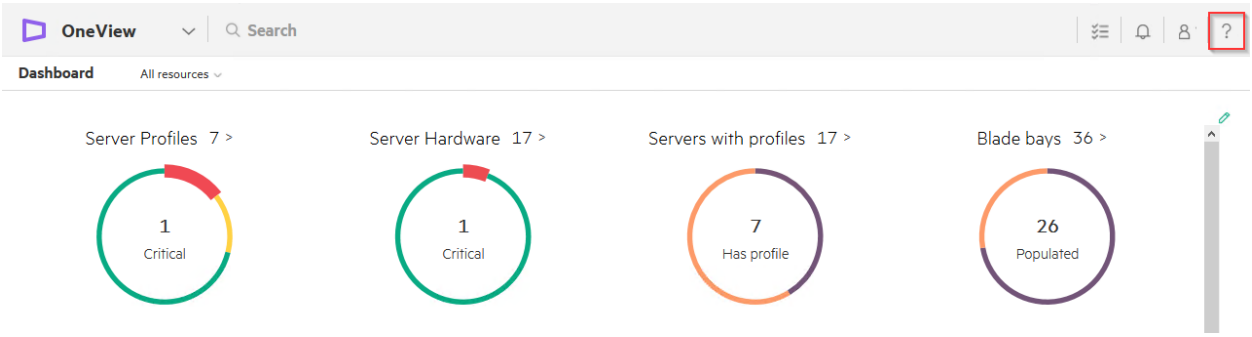
Login to the HPE Composer running OneView (https://composer) using the following credentials:

- Directory: **lj.lab**
- Username: **teamx**
- Password : **Passwordteamx**

with **x** being your team number



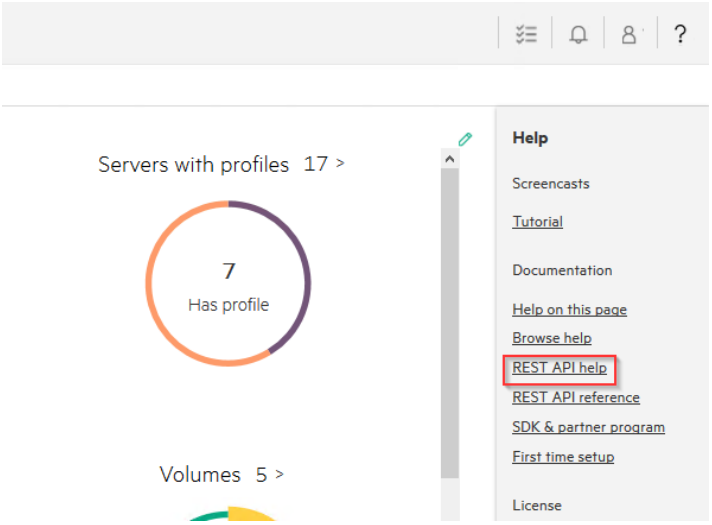
From the OneView interface, click on ? on the top right corner.



Three important links are available:

- REST API Help
- REST API reference
- SDK & Partner program

Let's start with the first one, click on **REST API Help**



The REST API help gathers all important information you need to know about our REST API.

HPE OneView REST API scripting help

The REST API scripting documentation is designed to work in tandem with the ***HPE OneView REST API Reference***.

The scripting scenarios list the sequence of REST API operations you must invoke for each task and include links into the ***HPE OneView REST API Reference*** to provide details about the resource model schema and JSON (JavaScript Object Notation) examples.

Topic	Where to find the information
Where can I learn about the REST interface?	Learning about REST APIs
What tasks can I perform using the REST APIs?	What can I do with HPE OneView REST APIs?
Where can I find detailed reference information about the REST API resource model schema and JSON examples?	***HPE OneView REST API Reference***
What tasks can I perform using the appliance UI?	**HPE OneView Help**
How do I subscribe to messages on the State-Change Message Bus?	Using the State-Change Message Bus (SCMB)
How do I subscribe to messages on the Metric Streaming Message Bus?	Using the Metric Streaming Message Bus (MSMB)
Where can I download code samples and a technical preview of PowerShell and Python libraries?	PowerShell and Python code sample libraries
Where can I find HPE OneView user guides and other manuals?	**http://www.hpe.com/info/synergy-docs**
How do I submit documentation feedback to Hewlett Packard Enterprise?	docsfeedback@hpe.com

Legal notices

To see the publication date, help system version and edition, and copyright and legal information, see the legal notices.

A OneView's Help is also available online at http://www.hpe.com/info/oneview/docs. There, you will find a downloadable version of the document, which is nice to have on your laptop as it presents itself as a set of HTML pages that is easy to navigate.

User Guides (26)	Type	Size	Date
Previous Versions of HPE OneView Documents on HPE Support Center	HTML		
HPE OneView 4.1 Help	HTML	N/A	Jun 2018
HPE OneView 4.1 Help for HPE Synergy	HTML	N/A	Jun 2018
HPE OneView 4.1 API Reference (API version 800)	HTML	N/A	Jun 2018

Back to the OneView UI, the first *Where to find information* link is a full section of the OneView help dedicated to learning about REST APIs.

HPE OneView REST API scripting help

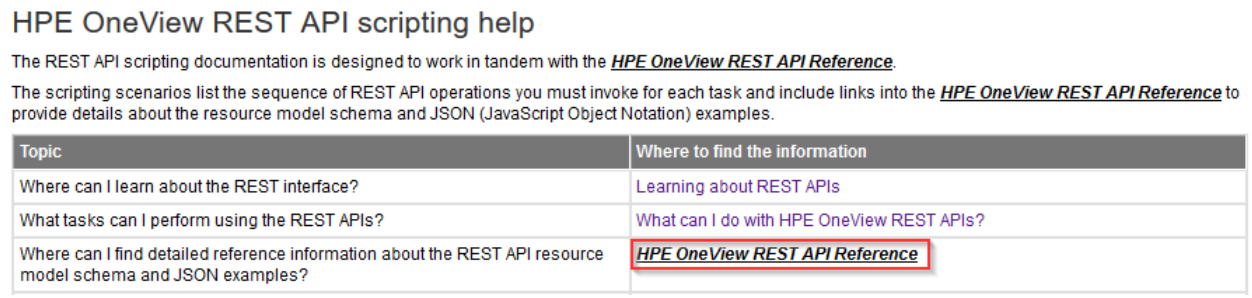
The REST API scripting documentation is designed to work in tandem with the ***HPE OneView REST API Reference***.

The scripting scenarios list the sequence of REST API operations you must invoke for each task and include links into the ***HPE OneView REST API Reference*** to provide details about the resource model schema and JSON (JavaScript Object Notation) examples.

Topic	Where to find the information
Where can I learn about the REST interface?	Learning about REST APIs
What tasks can I perform using the REST APIs?	What can I do with HPE OneView REST APIs?
Where can I find detailed reference information about the REST API resource model schema and JSON examples?	***HPE OneView REST API Reference***

Our lab will cover much of these general REST API concepts so take just a few minutes to explore the Learning section by clicking on **Learning about REST APIs**

REST APIs are an architectural style following common characteristics and principles; they don't all follow the same standard or specification. Users really have to read the documentation to understand how to use the API.
This detailed documentation is found in the *HPE OneView REST API Reference*



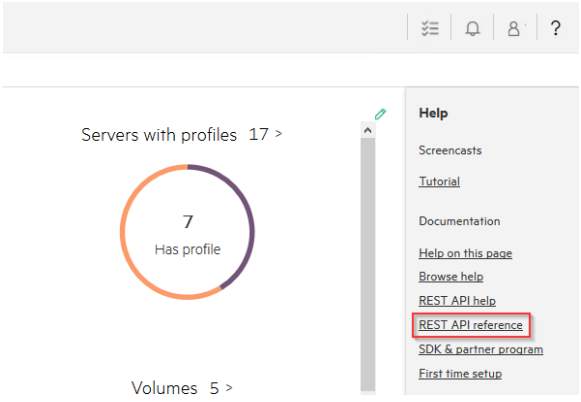
HPE OneView REST API scripting help

The REST API scripting documentation is designed to work in tandem with the ***HPE OneView REST API Reference***.

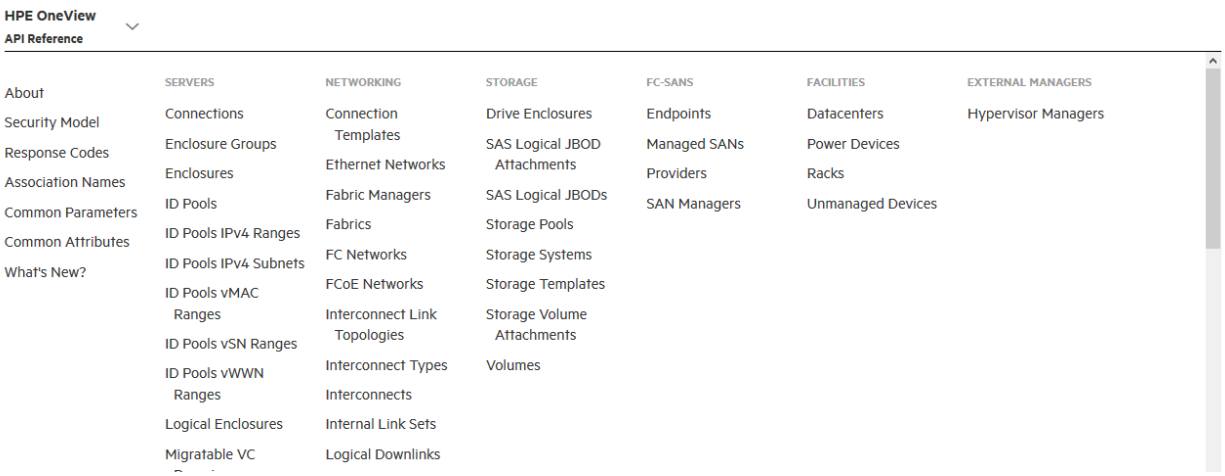
The scripting scenarios list the sequence of REST API operations you must invoke for each task and include links into the ***HPE OneView REST API Reference*** to provide details about the resource model schema and JSON (JavaScript Object Notation) examples.

Topic	Where to find the information
Where can I learn about the REST interface?	Learning about REST APIs
What tasks can I perform using the REST APIs?	What can I do with HPE OneView REST APIs?
Where can I find detailed reference information about the REST API resource model schema and JSON examples?	***HPE OneView REST API Reference***

The same link is also directly found on the main help menu:



Click on **HPE OneView REST API Reference**

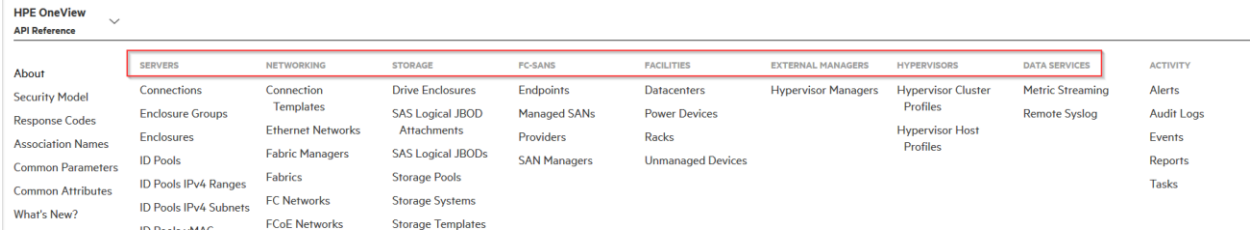


The Reference API document is where you can find detailed reference information about the REST API resource model schema and JSON examples.

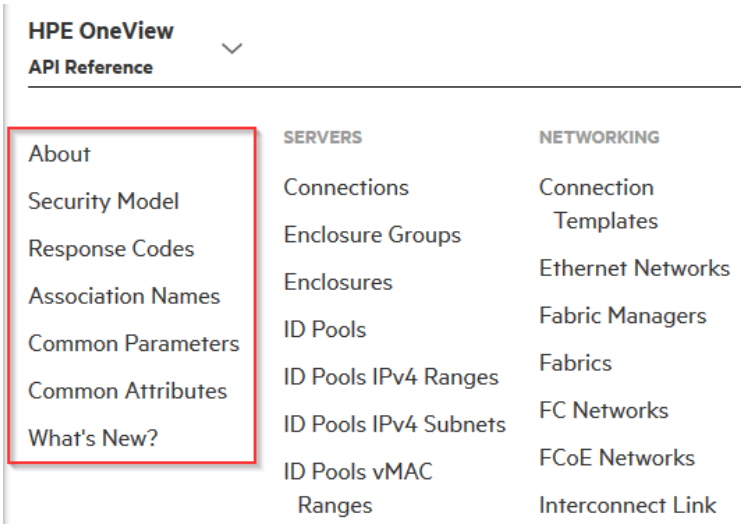
Just like for the general OneView help, the HPE OneView REST API Reference is also available online, a nice to have when you don't have access to a OneView console

User Guides (26)	Type	Size	Date
Previous Versions of HPE OneView Documents on HPE Support Center	HTML		
HPE OneView 4.1 Help	HTML	N/A	Jun 2018
HPE OneView 4.1 Help for HPE Synergy	HTML	N/A	Jun 2018
HPE OneView 4.1 API Reference (API version 800)	HTML	N/A	Jun 2018
HPE OneView 4.1 API Reference for HPE Synergy (API version 800)	HTML	N/A	Jun 2018
HPE OneView 4.1 User Guide	PDF	1.3 MB	Jun 2018

Back to the HPE OneView REST API Reference page, we can see that each single API call is regrouped in Component types (i.e. Servers, Networking, Storage, etc.) and Activity to ease the search. But you can also simply enter a resource name in the search pane to find what you need.

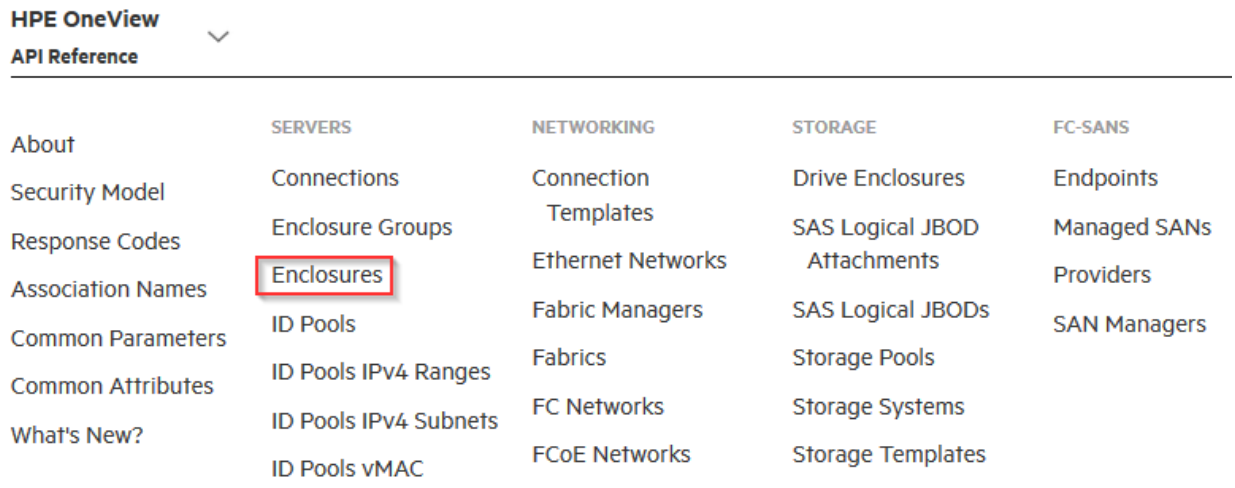


The group below provides important information about the API (supported versions), security model (authentication and authorization), associated names (association and relationships between resources), response codes (description of the different status codes), common parameters (how to create filters, count, sort, query), common attributes (description of the type attributes commonly used) and what's new.

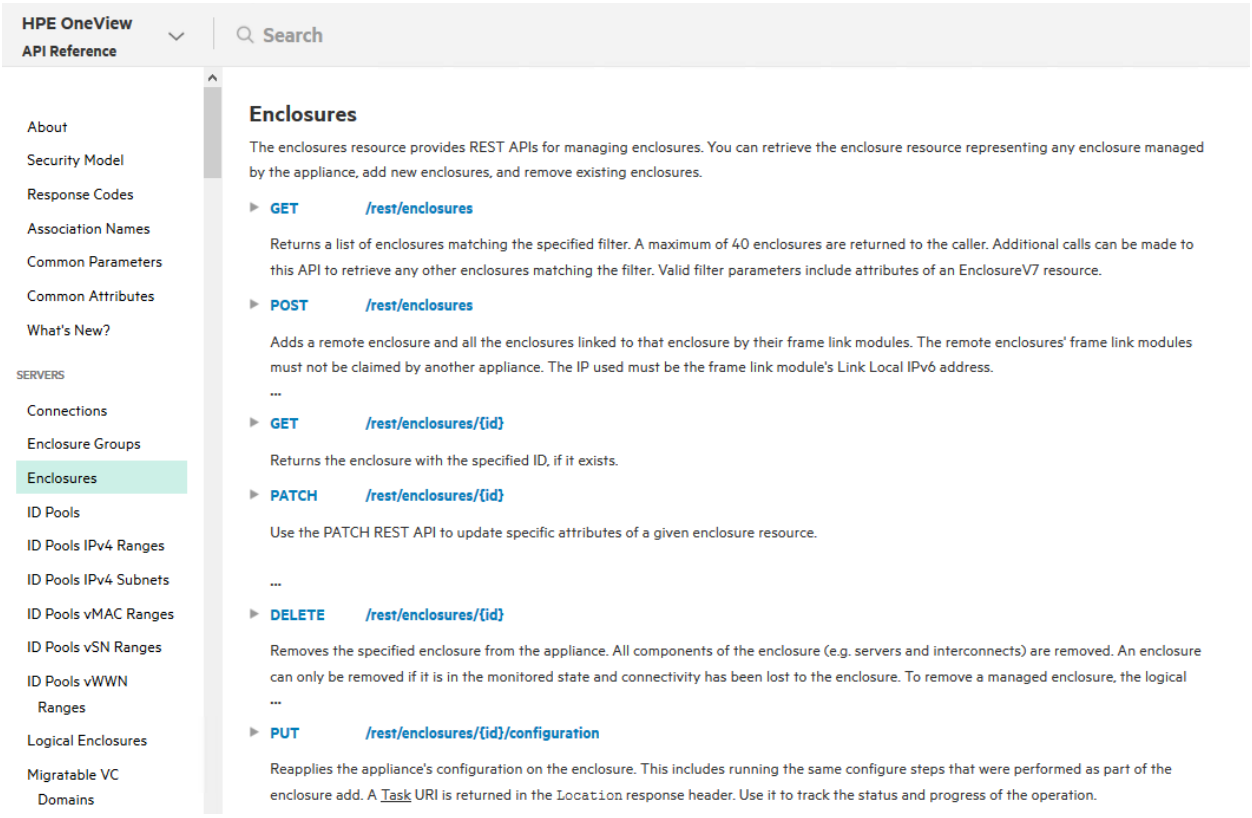


Navigate rapidly through the different menus to explore their content.

Click now on **Enclosure** as an example:



The following is displayed:

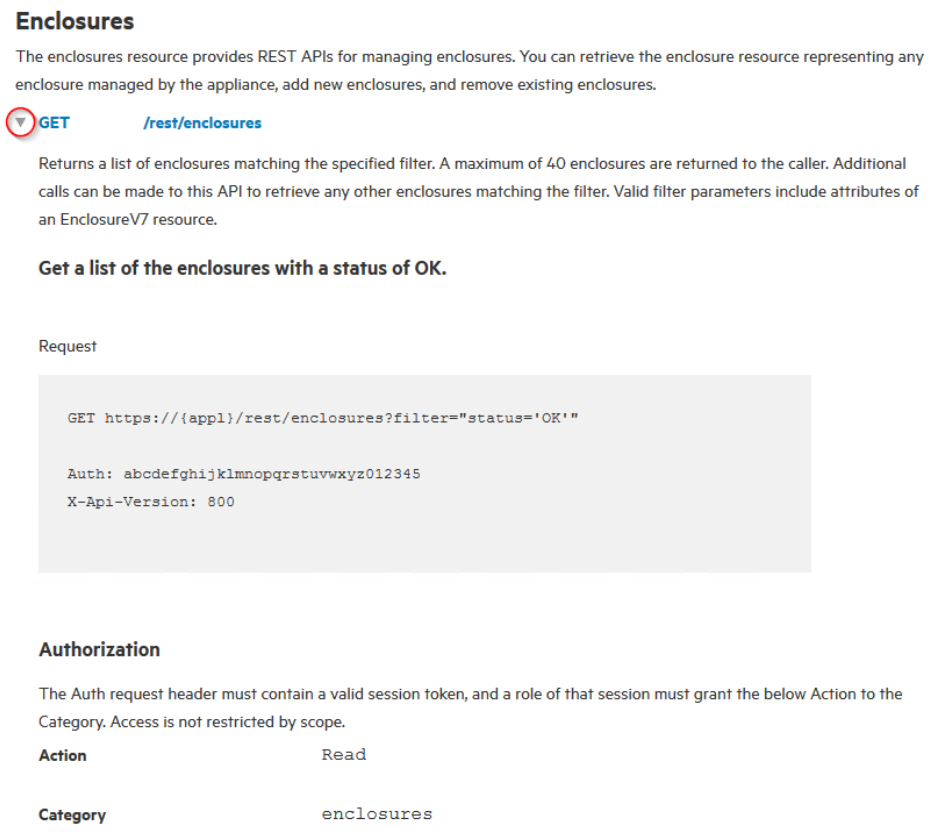


There is a page like this for each API resource method available in HPE OneView. An API resource method is an API invocation that mostly of the time contains a request and/or response property.

From the information provided, we can see all sorts of API invocation related to Enclosures, some `GET /rest/enclosures`, some `POST`, `PATCH`, `DELETE`, etc.

Each resource method performs a specific task. To get a full description of an invocation, you need to click on the expand icon.

Expand `GET /rest/enclosures`



Enclosures

The enclosures resource provides REST APIs for managing enclosures. You can retrieve the enclosure resource representing any enclosure managed by the appliance, add new enclosures, and remove existing enclosures.

GET **/rest/enclosures**

Returns a list of enclosures matching the specified filter. A maximum of 40 enclosures are returned to the caller. Additional calls can be made to this API to retrieve any other enclosures matching the filter. Valid filter parameters include attributes of an EnclosureV7 resource.

Get a list of the enclosures with a status of OK.

Request

```
GET https://{appl}/rest/enclosures?filter="status='OK'"

Auth: abcdefghijklmnopqrstuvwxyz012345
X-Api-Version: 800
```

Authorization

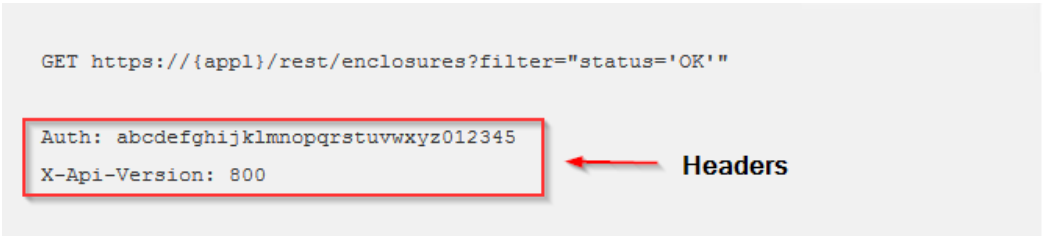
The Auth request header must contain a valid session token, and a role of that session must grant the below Action to the Category. Access is not restricted by scope.

Action Read

Category enclosures

What you get is always following the same pattern:

- **A description**: this invocation will return a list of enclosures.
- **One or more request examples**: showing example(s) of how to build invocations and what needs to be provided in the URL and in the headers:



With POST/PATCH/PUT invocations, headers and body are usually required:

```
POST https://{appl}/rest/enclosures

Content-Type: application/json
Auth: abcdefghijklmnopqrstuvwxyz012345
X-Api-Version: 800

{
   "hostname" : "fe80::1:2:3:4",
}
```

Headers

Body

- A **request headers section**: listing all the requested headers that have to be provided:

Request Headers

`Accept-Language`	The language code requested in the response. If a suitable match to the requested language is not available, en-US or the appliance locale is used.	string
`Auth`	Session authorization token obtained from logging in. If this header is not included or if the session-token is invalid, the response code will be 401 Unauthorized.	string required
`X-Api-Version`	Specifies the version of the API to invoke. The behavior of a given API version remains the same. It is upward compatible from release to release. New versions of an API are created when new features or changes are introduced. To ensure expected behavior, always provide the X-Api-Version value.	integer required

- Then depending of the type of invocation we have:
  - With GET:
    - **Query parameters**: describing the different query options: start, sort, count, fields, filter, query, etc.:

Query Parameters

`count`	The number of resources to return. A count of -1 requests all the items. The actual number of items in the response can differ from the requested count if the sum of start and count exceed the total number of items, or if returning the requested number of items would take too long. See the Common Parameters page for more information on the usage of count. Default -1	integer
`fields`	Specifies which fields should be returned in the result set.	string
`filter`	A general filter/query string to narrow the list of items returned. The default is no filter - all resources are returned. See the Common Parameters page for more information on the usage of filter.	array of string
`query`	A general query string to narrow the list of resources returned. The default is no query - all resources are returned. See the Common Parameters page for more information on the usage of query.	string

    - **Response body:** describing all the resource members that can be found in the response body:

Response Body	EnclosureListV7	
`category`	Identifies the resource type	string read only
`count`	The actual number of resources returned in the specified page	integer
`created`	Date and time when the resource was created Format yyyy-MM-dd'T'HH:mm:ss.SSS'Z' Pattern [1-2][0-9][0-9][0-9]-([0-1][0-9])-[0-3][0-9]T[0-2][0-9]:[0-5](.[0-9][0-9][0-9])?Z	timestamp read only
`eTag`	Entity tag/version ID of the resource, the same value that is returned in the ETag header on a GET of the resource	string read only
`members` array of EnclosureV7	List of enclosures resources.	
`applianceBayCount`	The number of appliance bays in the enclosure.	integer read only
`applianceBays`	A list of the appliance bays in the enclosure.	Appliance read only
`bayNumber`	The bay number of the appliance.	integer read only
`bayPowerState`	The power state of the appliance bay.	

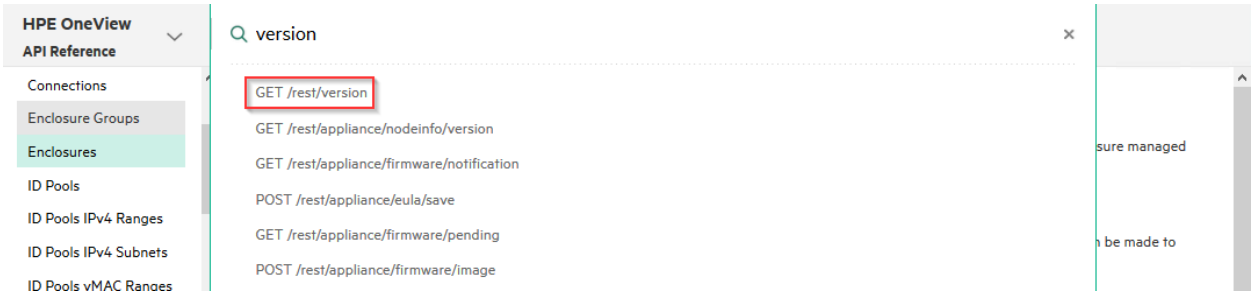
- With POST/PUT/PATCH/DELETE:
  - **Response header:** providing the newly created resource or task created to perform the requested action.

- With POST/PUT/PATCH/DELETE:
  - **Request Body:** describing the content of the body that needs to be provided)

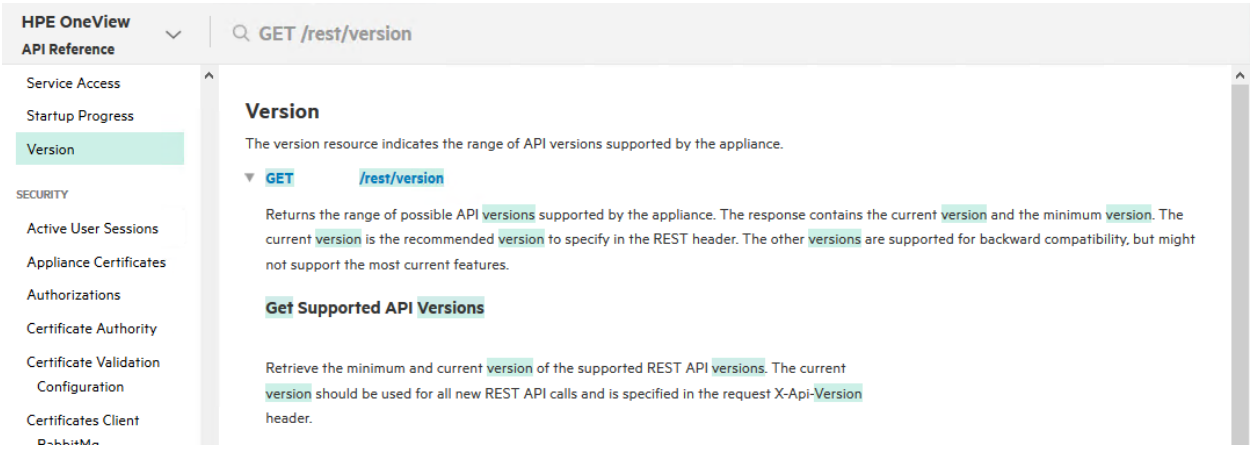
Request Body	array of PATCH Operation	
`from`	The JSON path to use as the source of a "move" or "copy" operation. Not used by "add", "remove", "replace", or "test".	string read only
`op`	The type of operation: one of "add", "copy", "move", "remove", "replace", or "test".	string required read only
	Default Operation	
`path`	The JSON path the operation is to use. The exact meaning depends on the type of operation.	string required
`value`	The value to add or replace for "add" and "replace" operations, or the value to compare against for a "test" operation. Not used by "copy", "move", or "remove".	any type read only

All this might sound very theoretical but don't worry, we are soon going to execute our first API call and you will get a much better understanding and fun.

As a first step toward the fun, enter **version** in the search pane then select `GET /rest/version`



You will find something like this:



From the information provided, we can see that a call to `GET /rest/version` will return the versions supported by the API. We will use this later for our first interaction with the HPE OneView API.

If you scroll down in the help page, you will also find what kind of response is expected:

Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "currentVersion" : 800,
  "minimumVersion" : 1
}
```

We can see, that this API returns a Content-Type of JSON (JavaScript Object Notation). This is a very popular format used to exchange data with an API. It is similar to XML, just a little more human readable.

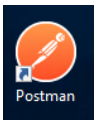
Many APIs are moving away from XML as the exchange format in preference of JSON. HPE OneView uses JSON for input parameters, and responses. There are many web sites for you to learn about JSON, but I have found this online parser very helpful to check the syntax of a JSON payload before using it: http://json.parser.online.fr/

Lab 2: REST Tool

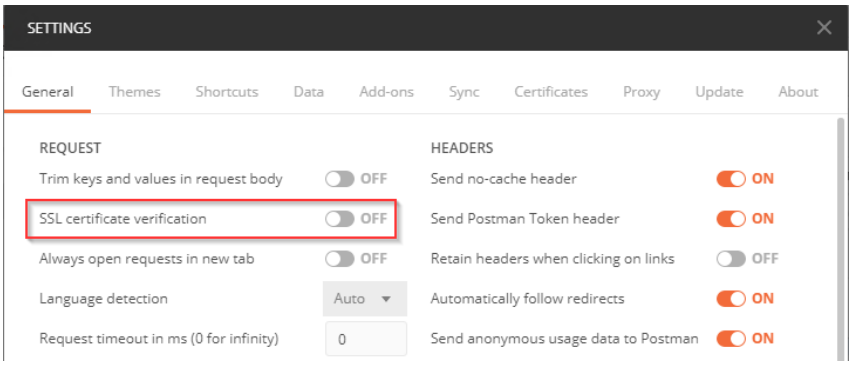
So now you have all the information you need to place your first HPE OneView API call to retrieve the version numbers. However, if you are new to REST programming, you might ask yourself: how do I place a REST call to an API without writing any code?
There are a number of simple solutions for this. My favorite is Postman but there are also several browser plug-ins available. You can install Postman as a Native App or a Chrome Extension.
Postman is one of the most complete REST tools and offers really useful features like:

- You can save your REST calls and share the collections with others
- You can use variables to store for instance the OneView authentication session keys

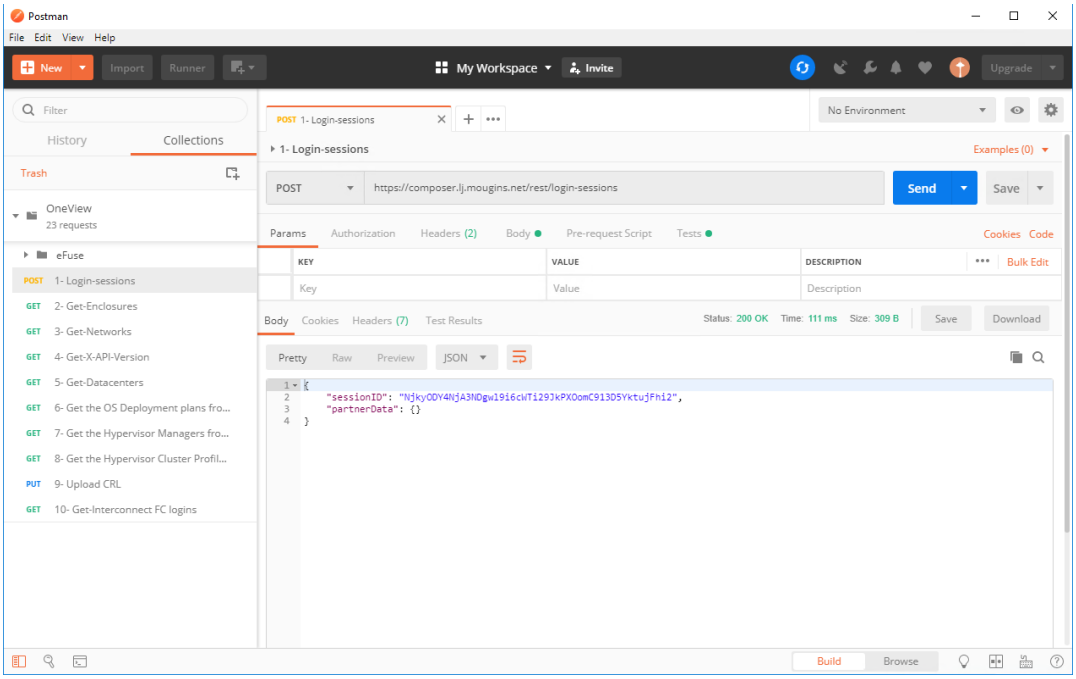
Open **Postman** using the shortcut on your desktop:



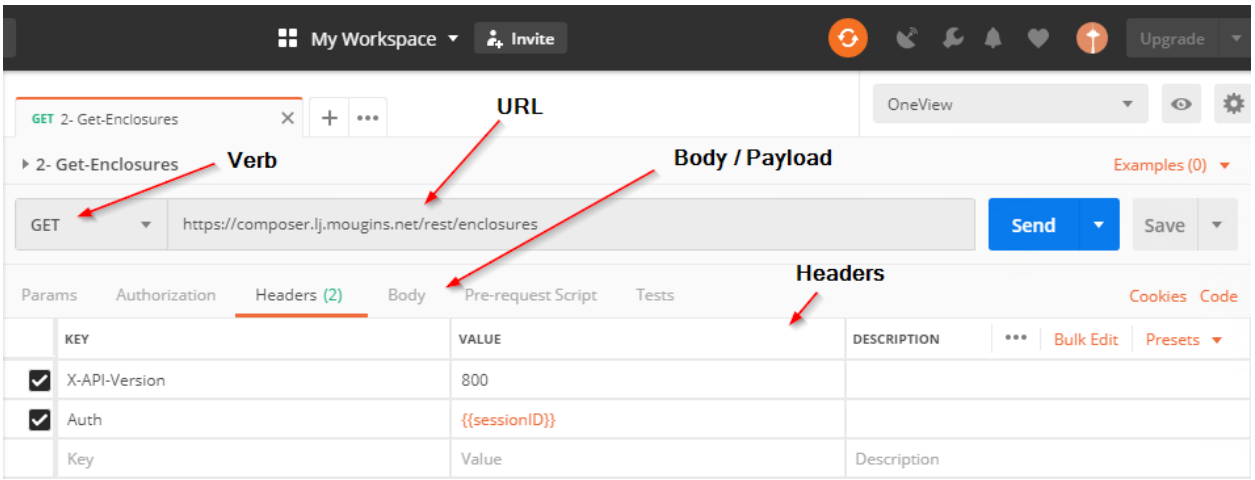
Note: Postman can be downloaded from https://dl.pstmn.io/download/latest/win?arch=64
Note: In Settings, you must set **SSL certificate verification** to **OFF** if OneView uses a self-signed certificate



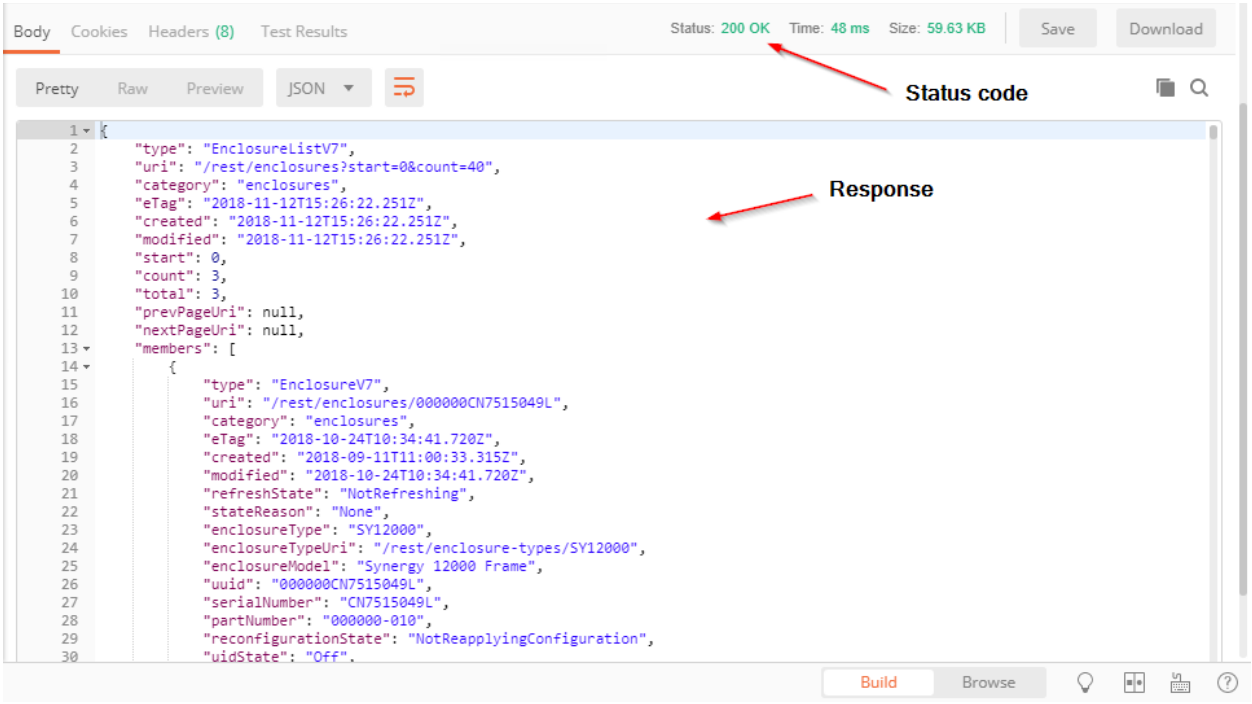
The following window opens:



Postman main pane provides the following:



The bottom pane is where the result of REST calls resides:



Let's make a try with `GET /rest/version` we saw previously.

If you go back to the *HPE OneView API Reference* documentation, you will find in the help what type of request is required to pass the `GET /rest/version` (returns the range of possible API versions supported by the appliance):

Request

```
GET https://{appl}/rest/version

Accept-Language: en_US
```

So, in more details, the documentation provides the following:

Field	Value
URL	`https://{appl}/rest/version`
Verb	`GET`
Header	`Accept-Language: en_US`

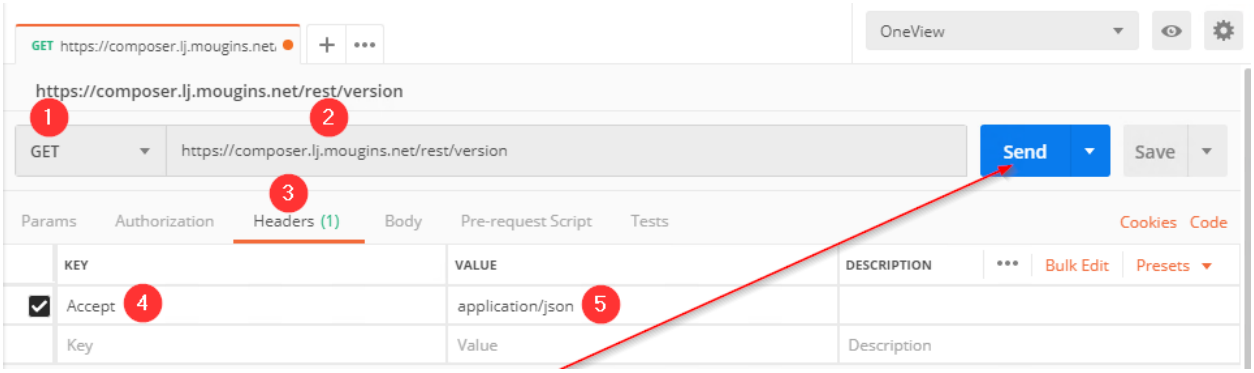
`Accept-Language: en_US` in the HTTP Header is optional. It just advertises which languages the client is able to understand so it will work without it.

Optionally, you can specify in the header `Accept=application/json` to explicitly tell the API that we expect a response in JSON. You might need to do that sometimes to avoid an XML response.

You can note that this request does not require `Auth` (any authentication) or `X-Api-Version` headers. So, there is no request payload to worry about for the `GET /rest/version` call

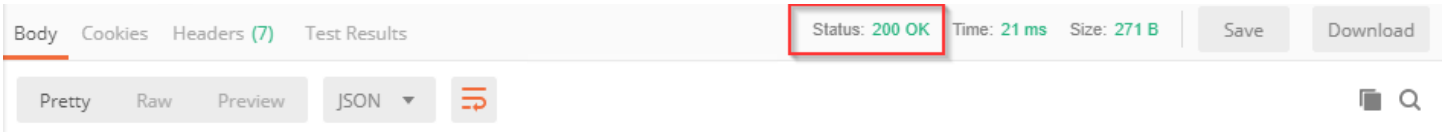
Enter in Postman the following information:

- In ① Select in Verb: **Get**
- In ② Enter the URL: **https://composer.lj.lab/rest/version**
- In ③ Select **Headers**
- (optional) Enter in *key* ④: **Accept**
- (optional) Enter in *value* ⑤: **application/json**

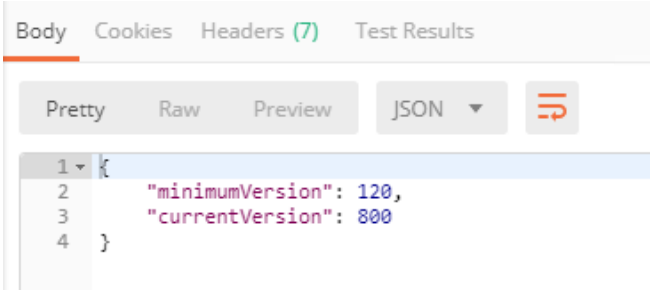


When ready, press the **Send** button.

Postman will contact the URL specified and invoke `GET version` from the API.
In the Response section, as illustrated below, you want to check the HTTP status code. A value of `200` means it was successful.



Next, you can check the Response body formatted as JSON, to find out about the version. This response will have to be parsed to retrieve the `currentVersion` value. But we can read pretty easily that `currentVersion=800` (meaning HPE OneView v4.1)



Congratulations! You have successfully placed your first call to the HPE OneView API. As you have seen, `GET version` does not require any authentication. In fact, it is one of the few calls you can place without having to provide proper authentication.

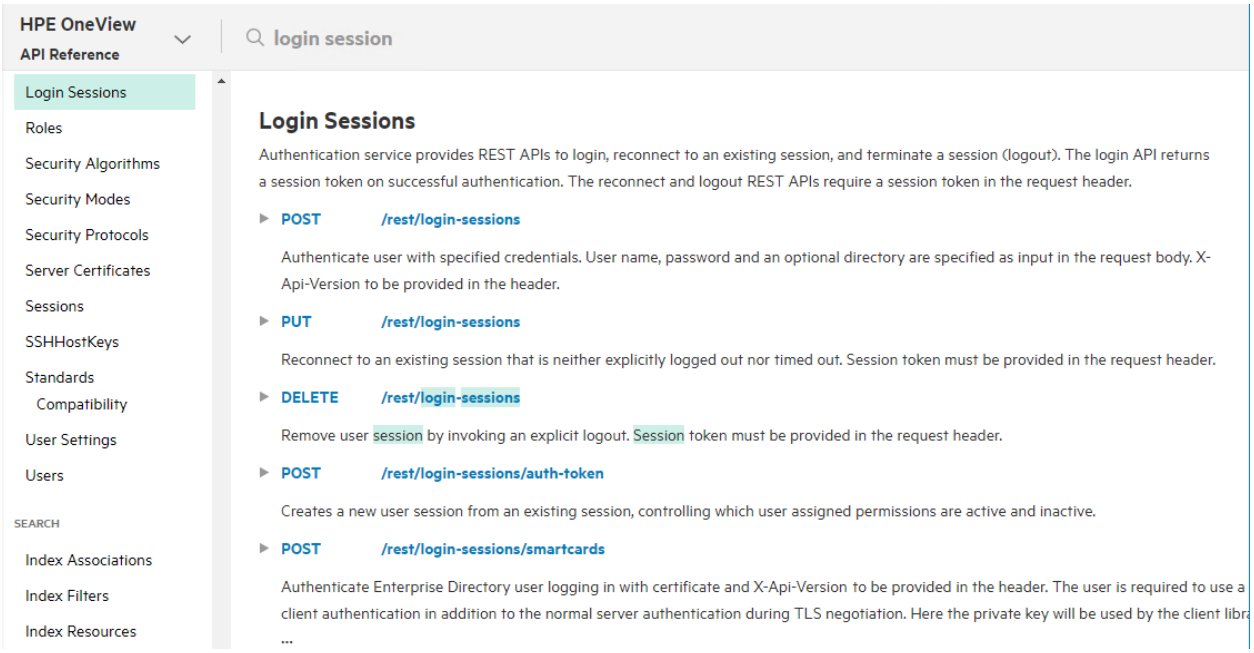
In subsequent sections, we will dive deeper into the API and learn about how to authenticate and retrieve information about the HPE OneView resources.

Although this example might seem very basic, any integration with HPE OneView should always start by querying the API versions supported, therefore, this is always going to be your very first step.

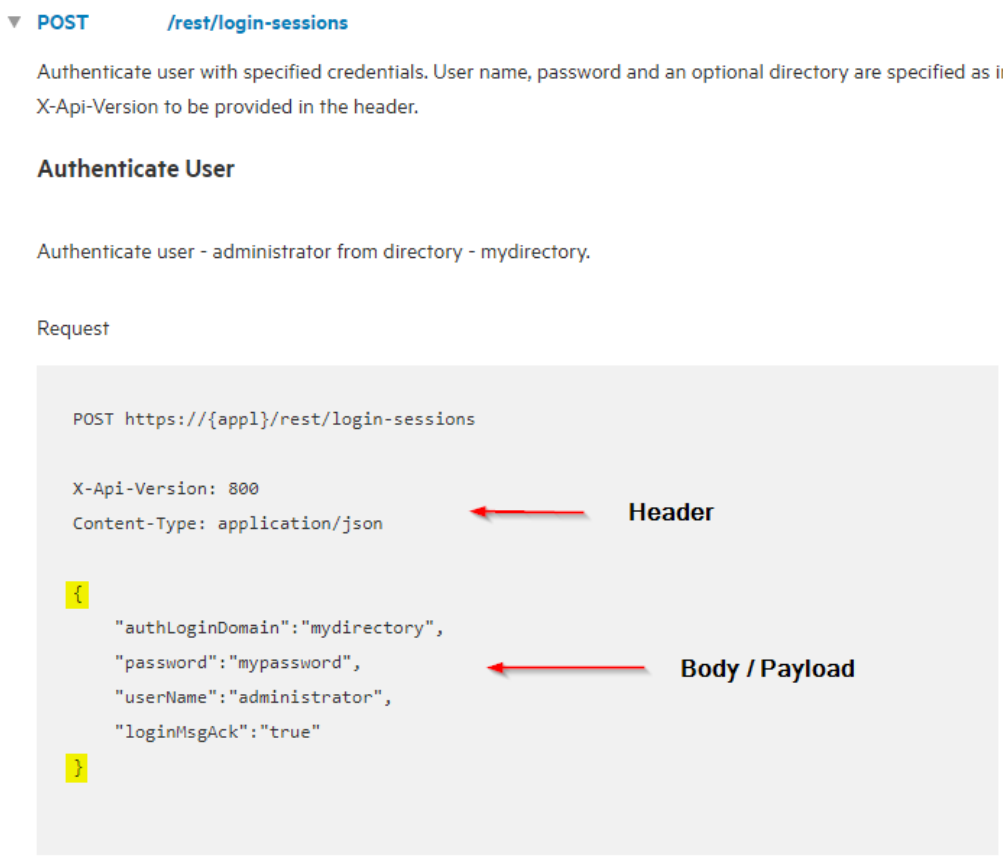
Lab 3: Login Sessions

In the previous section, we learned how to query the version of the HPE OneView API, using the `GET /rest/version` call. Let us now move forward and open a login session to HPE OneView. First, we can check the syntax of the login session by looking up **login** in the online help.

This is what we find:



If you expand the first POST, you have:



From the documentation, we can see that in order to open a login session, we need to use the `POST` method against the URI `/rest/login-sessions`. This will create (remember `POST` is the verb used in HTTP to create something) a session token, which we will use in the subsequent calls to the API.

We will ignore the login domain information for now.

We can also see in the documentation, that we will need to set two HTTP Headers:

- `X-API-Version`: set to the version of the API we want to use (let's assume `800`)
- `Content-Type`: set to `application/json`, to tell the API, that the payload we will be providing is in JSON notation

The payload expected by the API and notified by the `{ … }` must contains (at minimum) the following parameters:

- `userName`: set to the user name to use for authentication
- `password`: set to the password for that user name

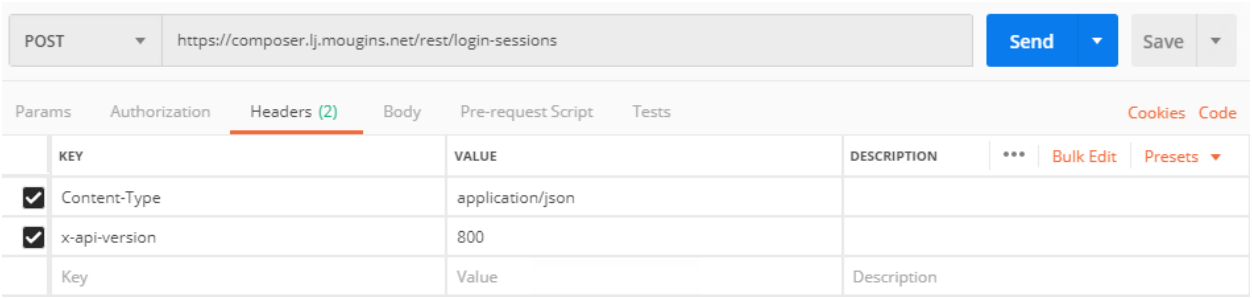
Note: Be careful, JSON is case sensitive. userName is expected here, not username, nor UserName. As a rule, the API expects lower case words, but capitalized, after the first word, if there are more than one word.

You can test this now using Postman, using the following parameters:

Field	Value
URL	`https://{appl}/rest/login-sessions`
Verb	`POST`
Header	`Content-type:application/json` `X-API-Version:800`

Payload	`{ "authLoginDomain":"mydirectory",`
	`"password":"mypassword",`
	`"userName":"administrator",`
	`"loginMsgAck":"true"`
	`}`

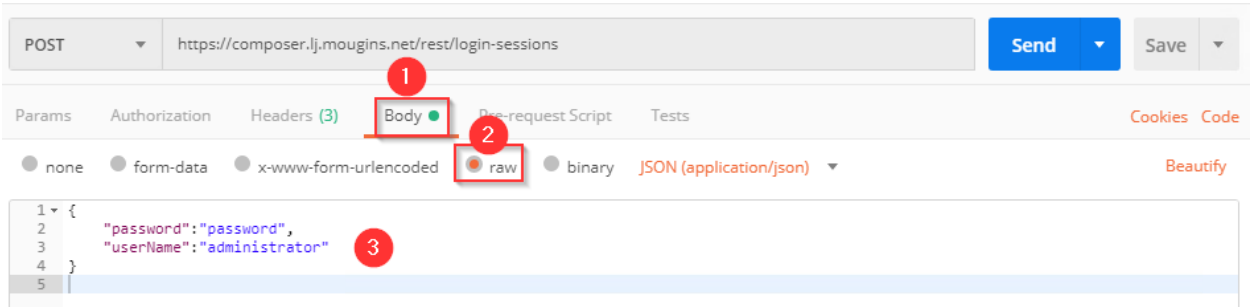
Content of the headers:



Content of the Body:

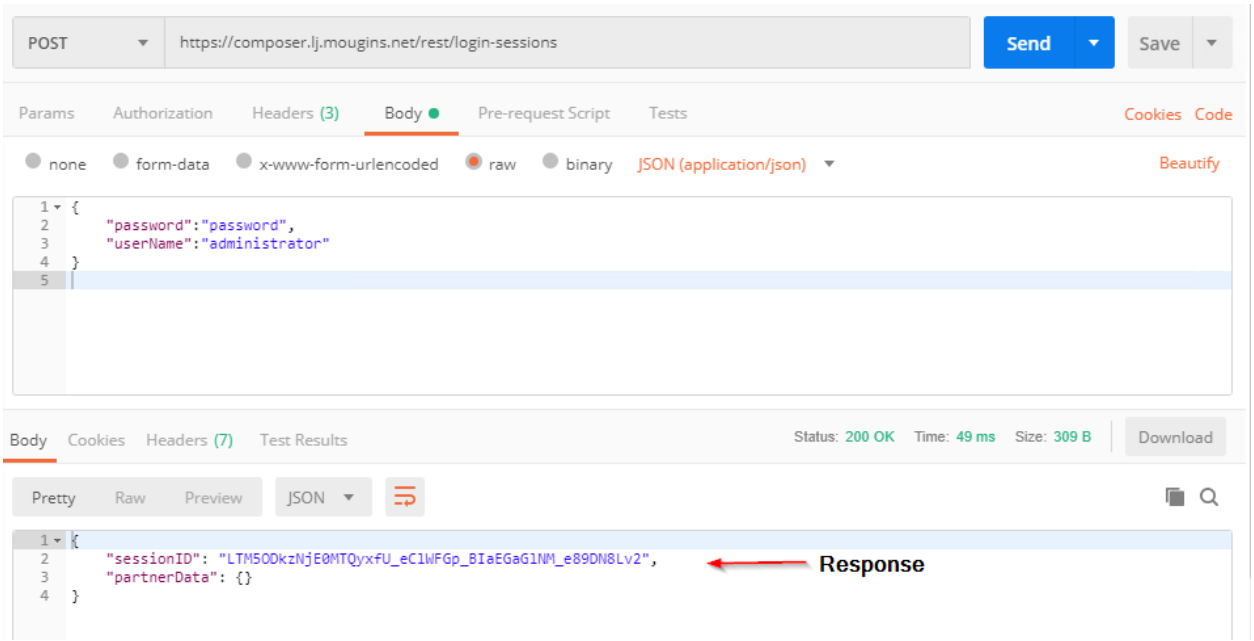
- Select **Body** in ①
- Select **raw** in ②
- Enter the payload in ③:

```
{
"password":"password",
"userName":"administrator"
}
```



Then press **Send**.

The response we get is the following:



The `sessionID`, also known as the session token is what we are looking for. This is the token that will be used for authentication in the subsequent calls to the API.
So, in order to be authorized to call any other API methods, we will need to provide the session token as another HTTP Header called `Auth`.

And remember, we now have two HTTP Headers, which have to be provided at each call to the API: `Auth` and `X-API-Version`.

Note: You should also be aware that session tokens expire 24 hours after the last utilization.

We can now test this token and call another API method. Let us pick `GET /rest/global-settings` which returns the current settings of the HPE OneView appliance.

Back to the API documentation, enter **global** and select `GET /rest/global-settings`

This is what we can read:

Global Settings

The global settings resource provides APIs and a place to write and read global settings. The settings service is used to store and retrieve name/value pairs which can be read and written by any service in the stack.

▼ **GET** **/rest/global-settings**

Gets a list of settings for all the authorized categories based on optional sorting & filtering and constrained by start and count parameters

List Settings

This example returns the list of settings, starting with index 0, and with a maximum count of 1

Request

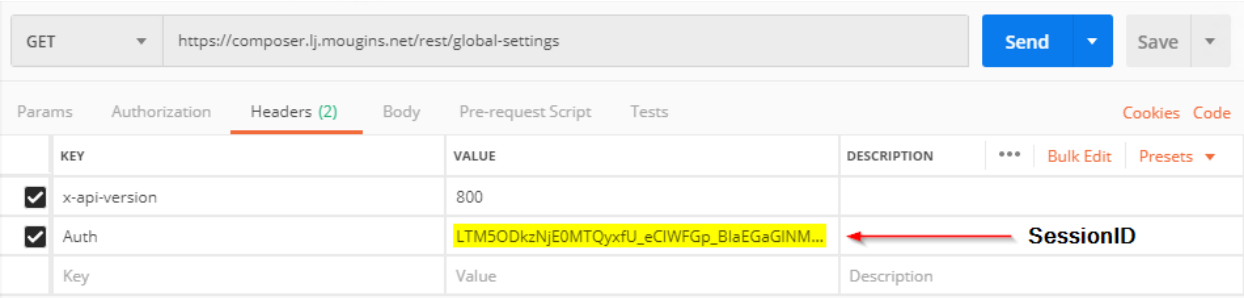
```
GET https://{appl}/rest/global-settings?start=0&count=1

Auth: abcdefghijklmnopqrstuvwxyz012345
X-Api-Version: 800
```

Notice now the presence of `Auth` with the `sessionID` in the header for this call.

So use Postman with the parameters found here:

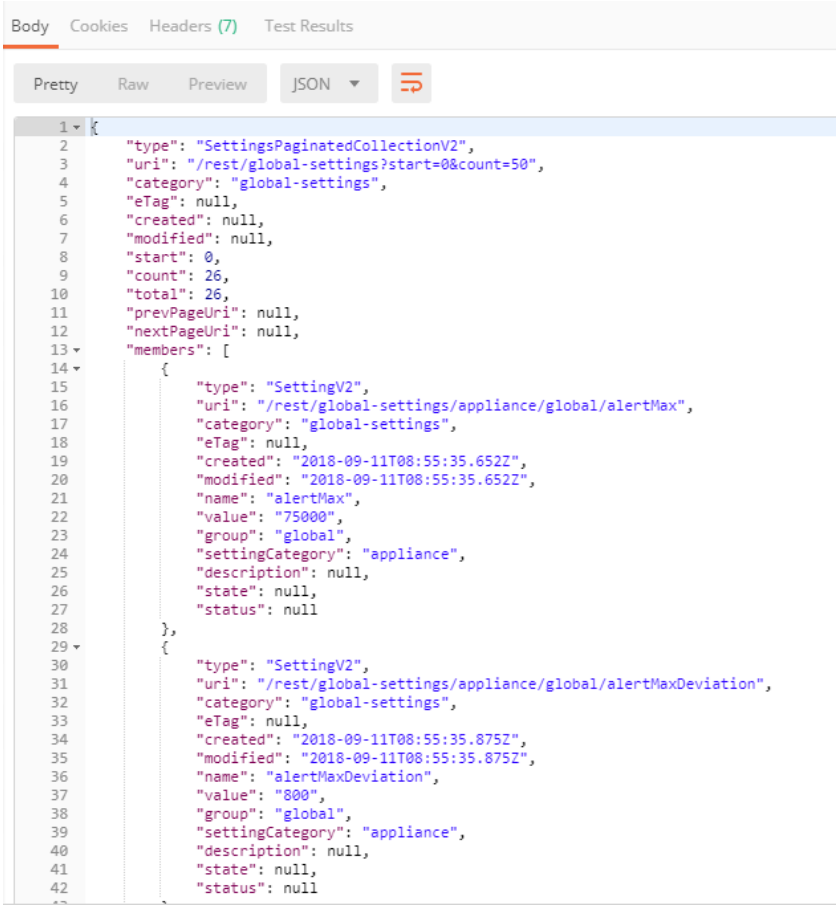
Field	**Value**
URL	`https://composer.lj.lab/rest/global-settings`
Verb	`GET`
Header	`Auth : <your sesionID>` `X-API-Version:800`

So, it is necessary to copy/paste your SessionID from your POST /rest/login-sessions in the auth value field:



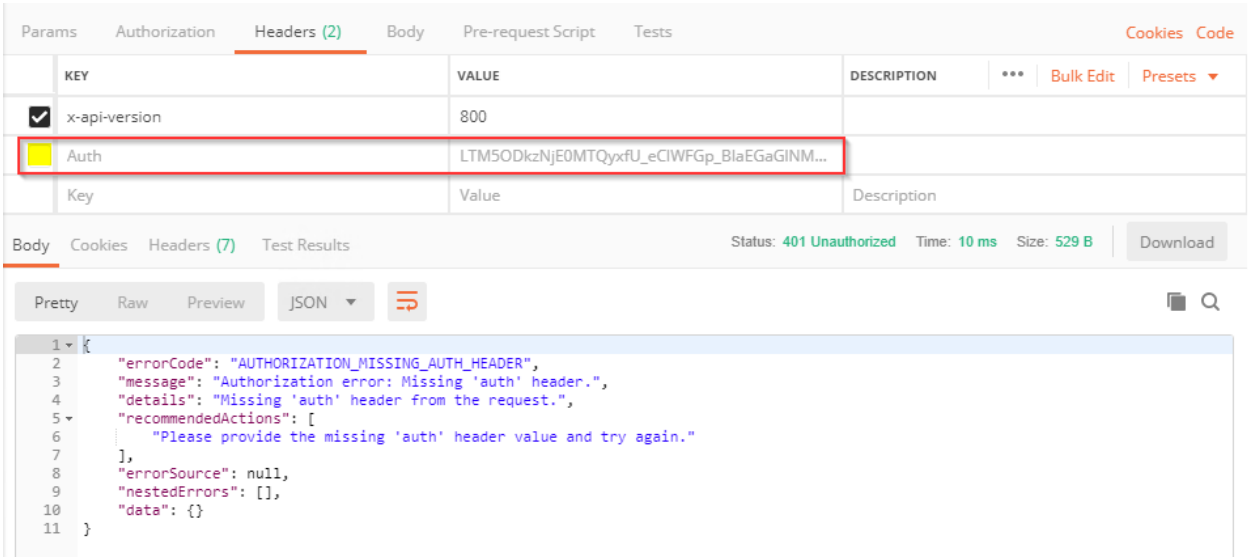
Then when all set, press **Send**

You should get the global settings information in the response body, again in a JSON format:




```
{
    "type": "SettingsPaginatedCollectionV2",
    "uri": "/rest/global-settings?start=0&count=50",
    "category": "global-settings",
    "eTag": null,
    "created": null,
    "modified": null,
    "start": 0,
    "count": 26,
    "total": 26,
    "prevPageUri": null,
    "nextPageUri": null,
    "members": [
        {
            "type": "SettingV2",
            "uri": "/rest/global-settings/appliance/global/alertMax",
            "category": "global-settings",
            "eTag": null,
            "created": "2018-09-11T08:55:35.652Z",
            "modified": "2018-09-11T08:55:35.652Z",
            "name": "alertMax",
            "value": "75000",
            "group": "global",
            "settingCategory": "appliance",
            "description": null,
            "state": null,
            "status": null
        },
        {
            "type": "SettingV2",
            "uri": "/rest/global-settings/appliance/global/alertMaxDeviation",
            "category": "global-settings",
            "eTag": null,
            "created": "2018-09-11T08:55:35.875Z",
            "modified": "2018-09-11T08:55:35.875Z",
            "name": "alertMaxDeviation",
            "value": "800",
            "group": "global",
            "settingCategory": "appliance",
            "description": null,
            "state": null,
            "status": null
```

As a quick check, remove the `Auth` HTTP Header (uncheck the option), and try again.



You should get a `401 Unauthorized Status` code. The details of the error clearly state that you forgot to provide an `Auth` header for that request.

Note: HTTP Headers are case insensitive: `Auth` or `auth` would work fine.

Lab 4: API Version and backward compatibility

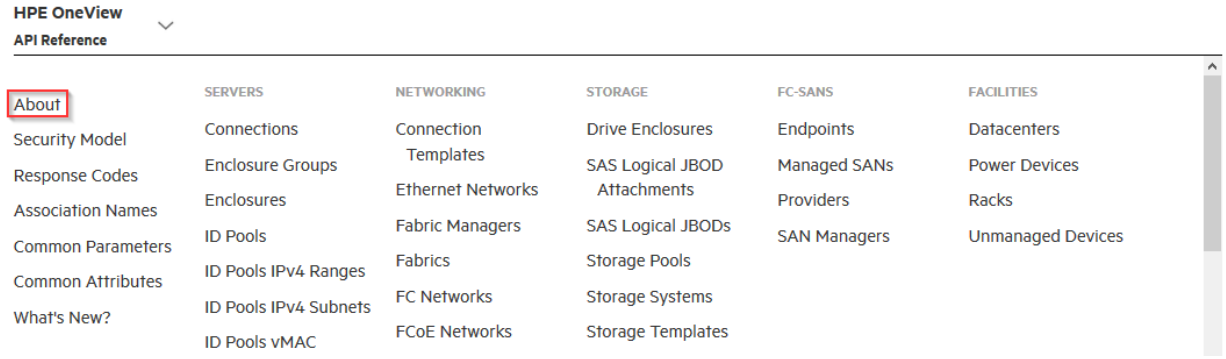
We discussed previously about how important it is to provide an API with software so that your ecosystem can grow and other software entities can integrate with it. But what happens when another release of the software comes up? New capability may be added, older behavior modified, which is normal, but it makes it more difficult to allow the software to evolve without disrupting the growing ecosystem. This is where backward compatibility becomes critical. We, as an API provider, need to guarantee that existing software integrations are not going to break when a new release is introduced.

In order to make this happen, API providers usually require integrators to specify the version of the API that they expect. In most REST APIs, this is done with a HTTP Header. And the API provider guarantees that older versions of the API remain unchanged. Of course, if an integrator wants to benefit from the newest functionality, it will have to update their software to use the newest API, but the older version, running against the older API, should continue to work without modification.

It is the API provider's decision to decide how many older versions of the API are supported at any given release and deprecate older ones if necessary.

This information is found in the *HPE OneView API Reference* documentation.

From the main menu, select **About**



Scroll down to **Supported REST API versions**

The HPE OneView API has been around for a few years already and the table below shows the versions of the API supported for each release:

HPE OneView release	REST API version
1.20	120
2.00	200
2.00.06	201
3.00	300
3.10	500
4.00	600
4.10	800

You can see that `800` is the API version for OneView 4.1.

The second table lists the API versions that are no longer supported in our release version (4.1)

HPE OneView release	REST API version
1.0, 1.01	3
1.05	4
1.10	101

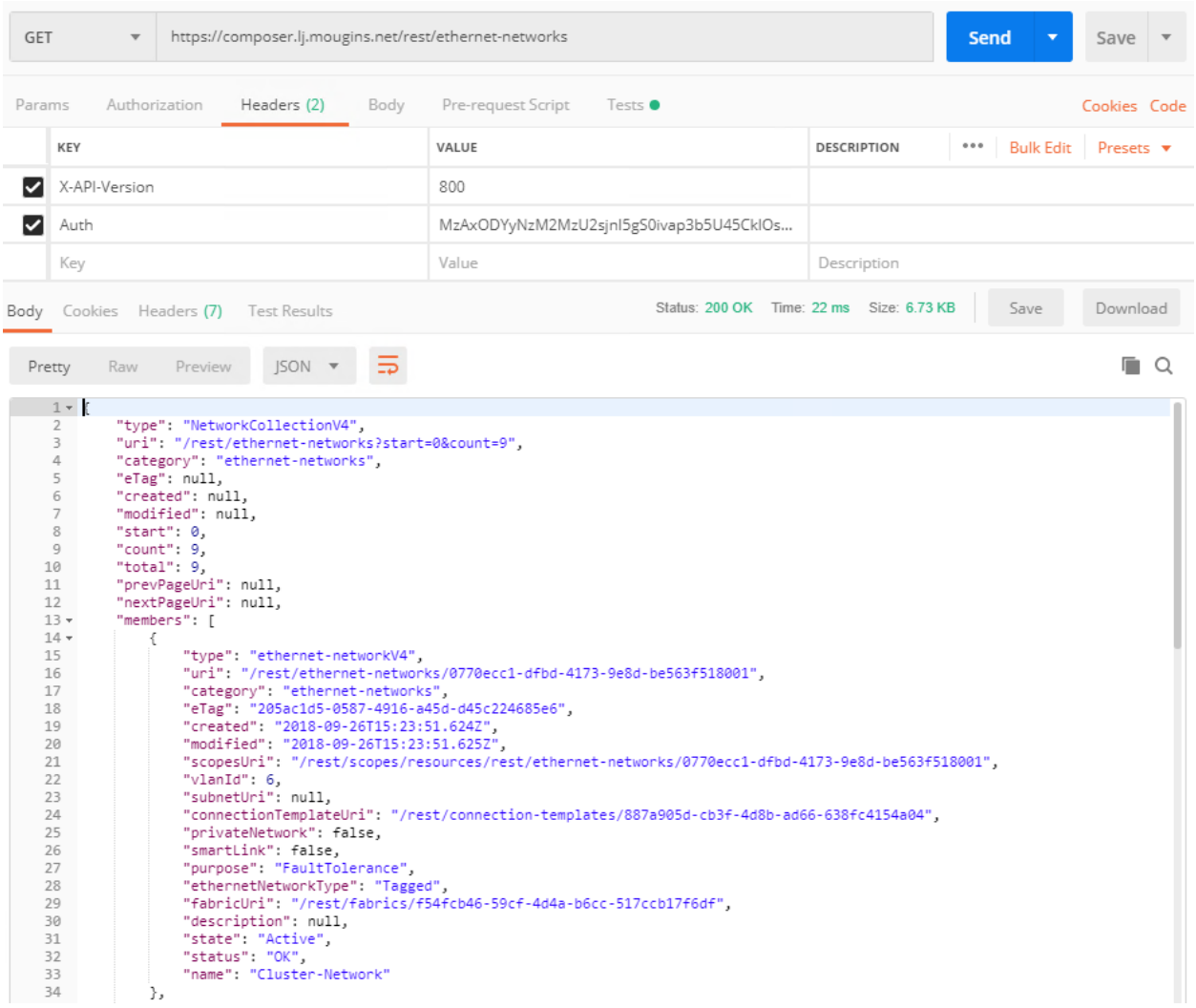
Any REST call using these API versions usually works but will not be supported.

In order to illustrate the impact of using a different API version, let us pick `GET /rest/ ethernet-networks` which returns the current Ethernet network present in the HPE OneView appliance.

So, use Postman with the following parameters:

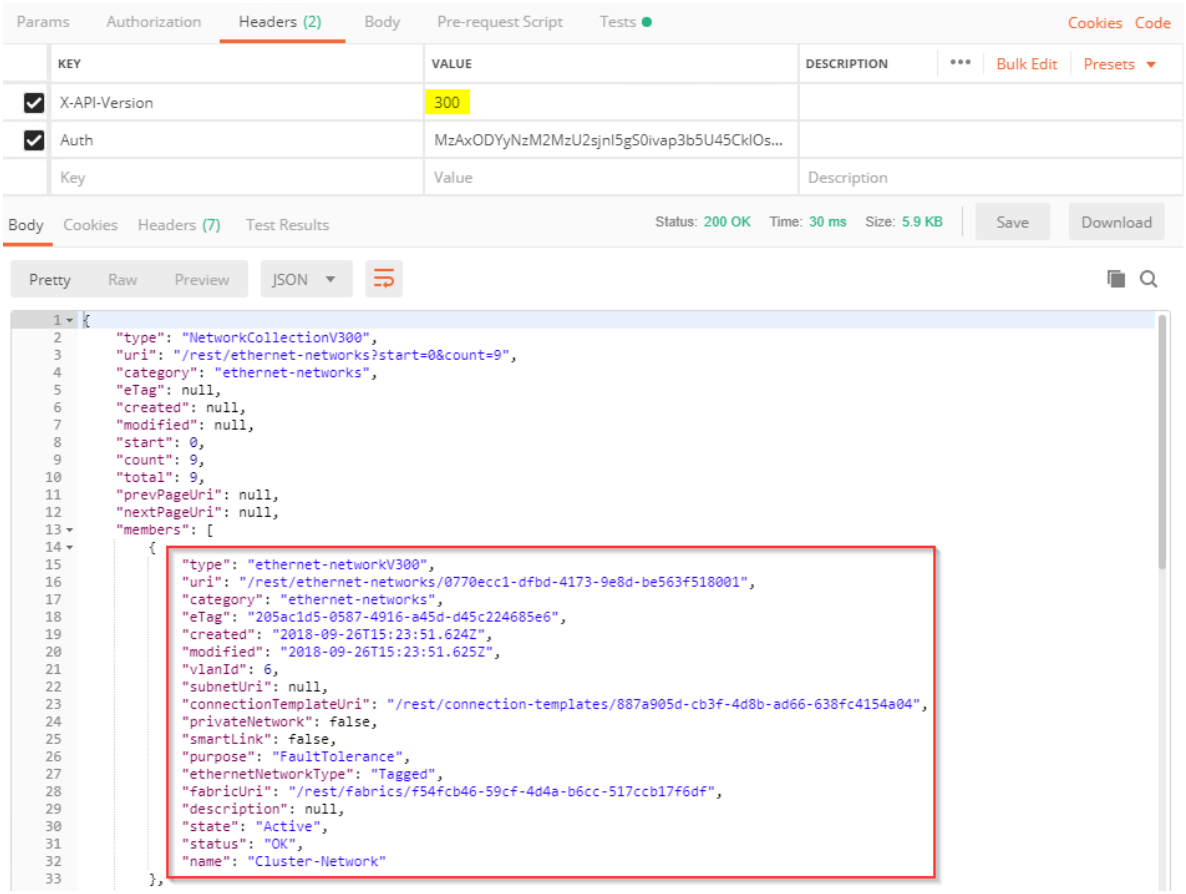
Field	Value
URL	`https://composer.lj.lab/rest/ethernet-networks`
Verb	`GET`
Header	`Auth : your sesionID` `X-API-Version:800`

The body response is the following:



Then place the same method (you can do a duplicate tab using a right-click) but this time using `X-API-Version:300`

We can see that the response in the body is slightly different than previously when using the version 800:



With version 800, Scope Based Access Control is present here as it was introduced in OneView 4.00 (REST API Version 600):


```
"type": "ethernet-networkV4",
"uri": "/rest/ethernet-networks/0770ecc1-dfbd-4173-9e8d-be563f518001",
"category": "ethernet-networks",
"eTag": "205ac1d5-0587-4916-a45d-d45c224685e6",
"created": "2018-09-26T15:23:51.624Z",
"modified": "2018-09-26T15:23:51.625Z",
"scopesUri": "/rest/scopes/resources/rest/ethernet-networks/0770ecc1-dfbd-4173-9e8d-be563f518001",
"vlanId": 6,
"subnetUri": null,
"connectionTemplateUri": "/rest/connection-templates/887a905d-cb3f-4d8b-ad66-638fc4154a04",
"privateNetwork": false,
"smartLink": false,
"purpose": "FaultTolerance",
"ethernetNetworkType": "Tagged",
"fabricUri": "/rest/fabrics/f54fcb46-59cf-4d4a-b6cc-517ccb17f6df",
"description": null,
"state": "Active",
"status": "OK",
"name": "Cluster-Network"
```


So basically, any version below 600 will never provide/support any scopes information.

Lab 5: API Resource model

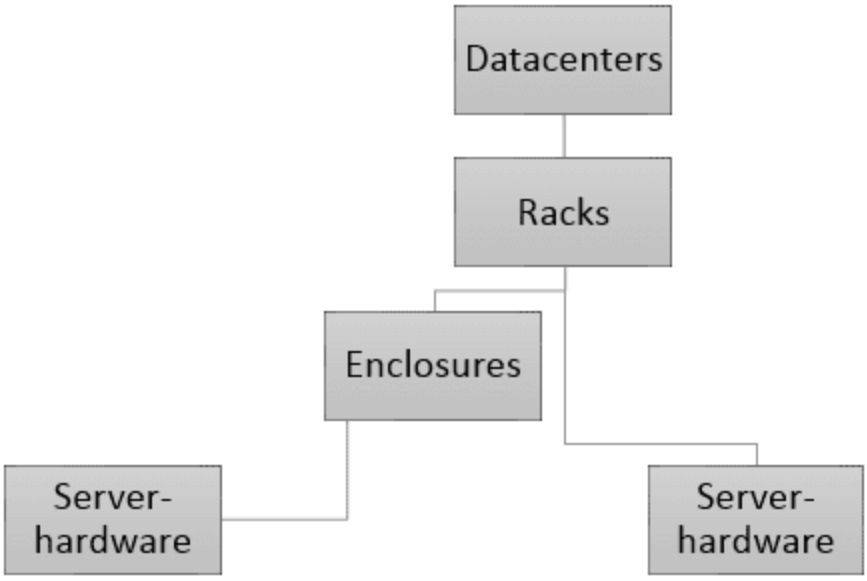
In previous labs, we saw some of the concept on the versioning, the authentication and the discovery of the infrastructure, via a REST API approach.

The next important subject that we need to understand is the API resource model.

HPE OneView provides a fully connected resource model with associations or more precisely relationships between the HPE OneView resources. These associations are fundamental during script developments.

So, let us discover the API resource model, in a top down approach through the objects available.

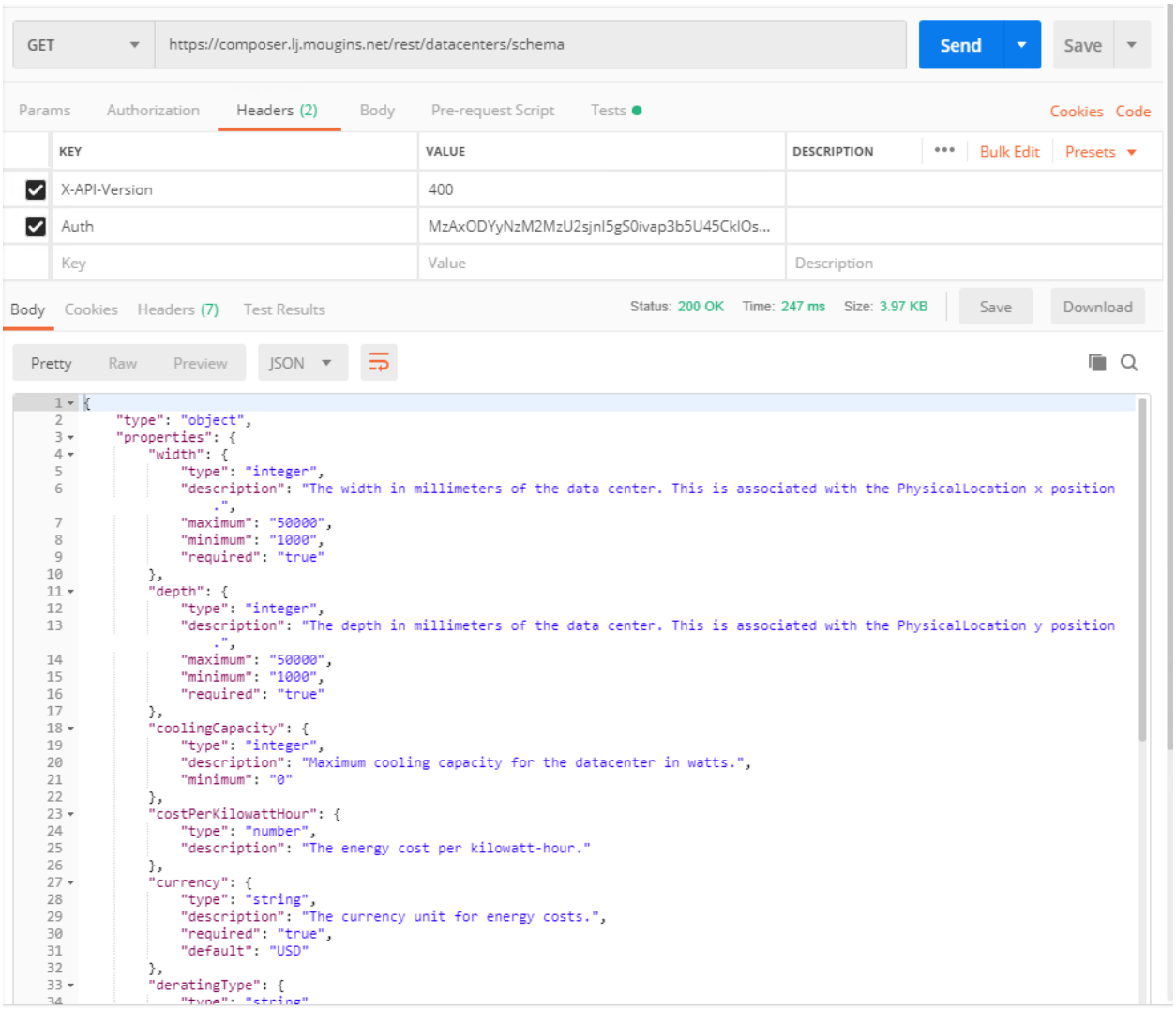
The following illustration shows the objects we will check during this process:



From the graph, we know that the top most object is a Datacenter, and we will use our REST client to browse for available Datacenters in our configuration using `/rest/datacenters`.

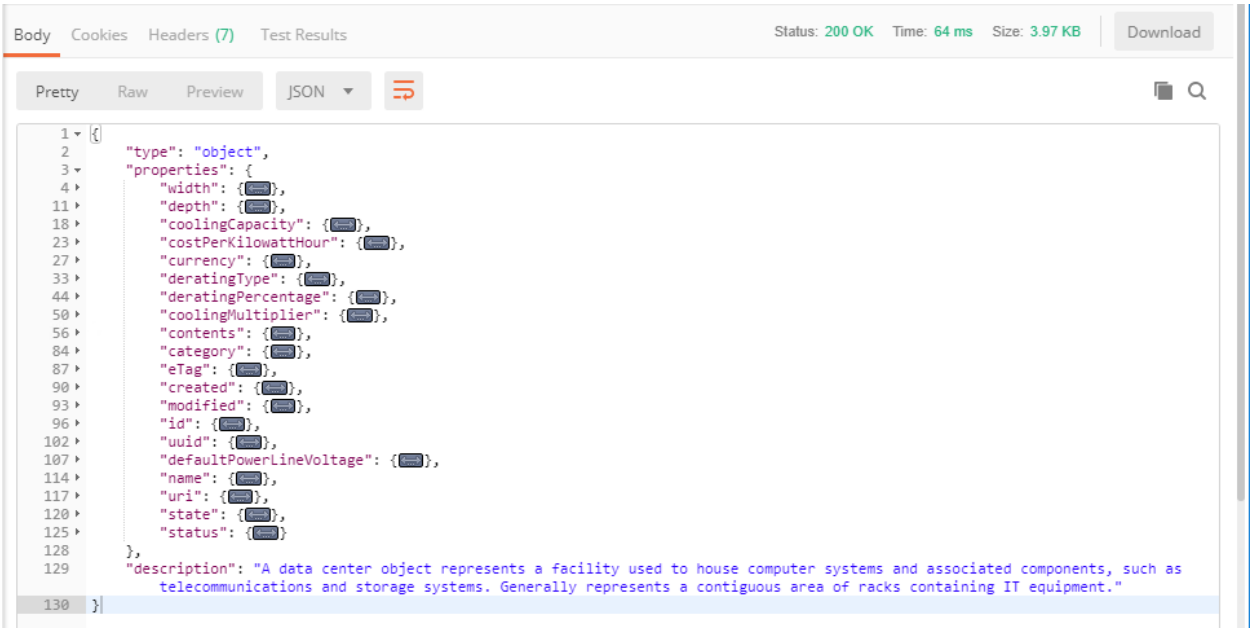
Before we do so, how do we know what are the properties that make up a Datacenter object?
There is a simple method to find out about this. This technique is to add a `/schema` at the end of the URL, for example `/rest/datacenters/schema`, to retrieve the schema for datacenter.

Let us try that with our Postman utility.

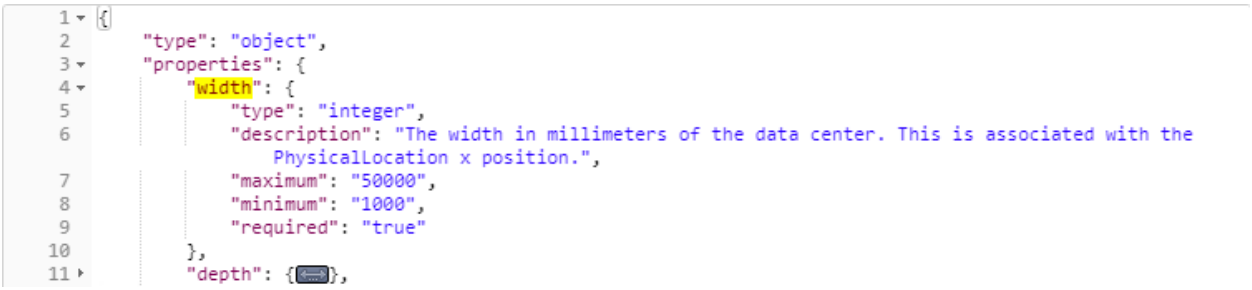


If you find the result difficult to read, you can collapse and expand the properties of an object and understand what is available.

For example, from the capture below, we can see that a Datacenter has a dozen properties such as width, depth, or status.

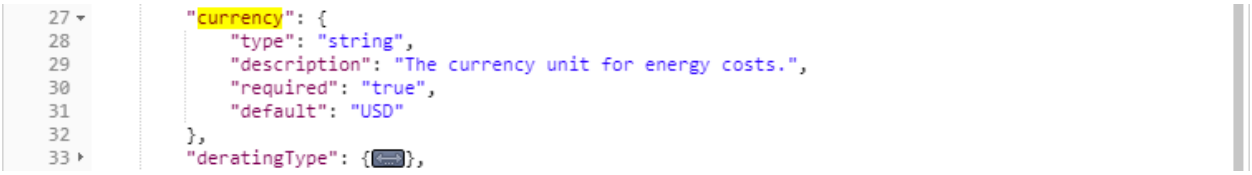


We can continue this exploration and drill down on **width**, which has a description, a type (*integer*), and a min and a max value.



We also can find a **required** flag, which is important, as required properties are mandatory when creating new objects with the `POST` method.

We can find other types of object. For example, expand the **currency** property and you will see that the type is *string*.



Integer and *string* are simple types, but if you continue to scroll down the list of properties, you will see the **contents** property, which is an *array* of items, of type *object*. An item is constructed from a number of properties such as *rotation*, *x*, *y* and a *resourceUri*. We also call this a *collection*.

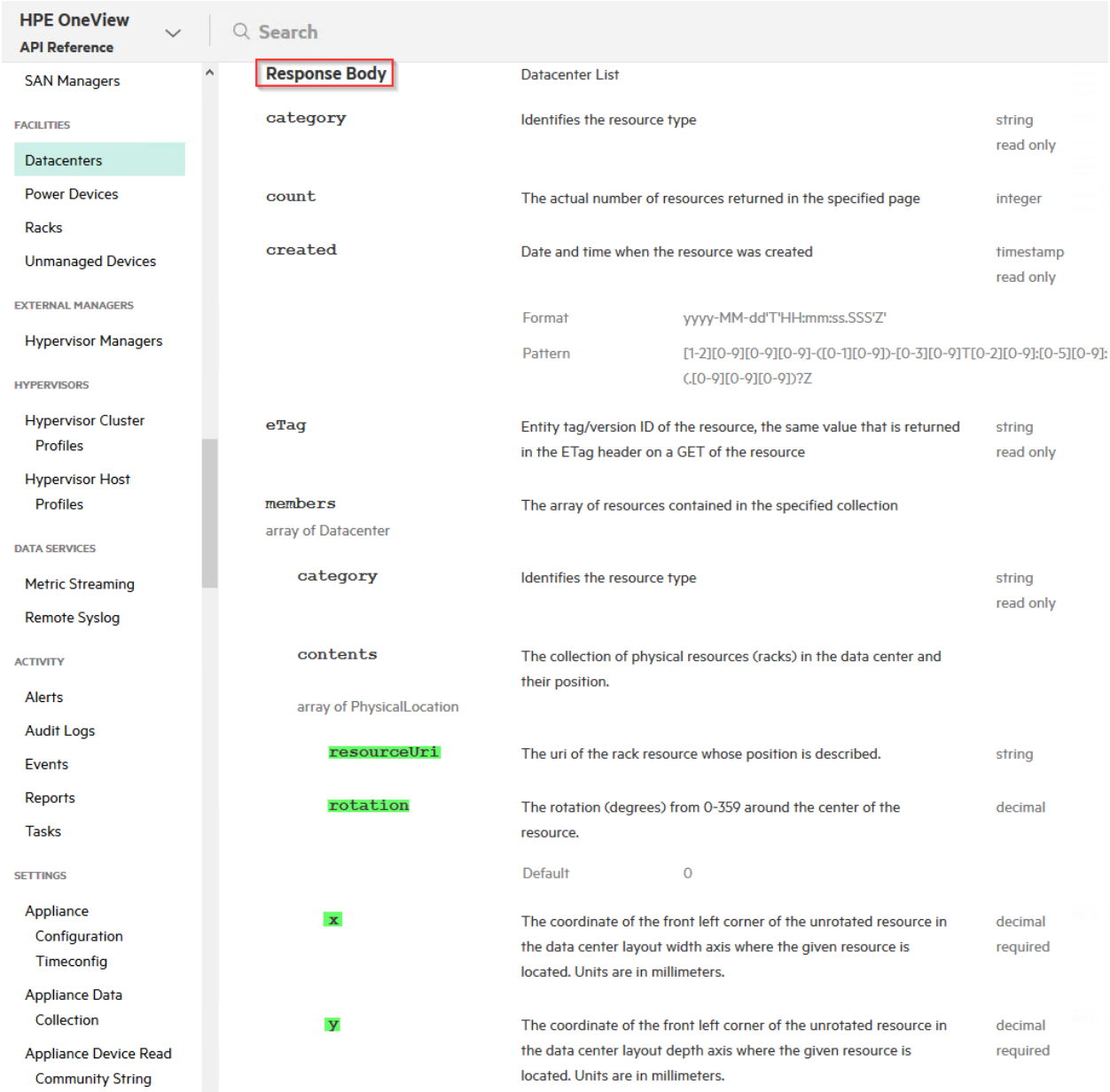
```
50      "coolingMultiplier": {...},
56      "contents": {
57          "type": "array",
58          "items": {
59              "type": "object",
60              "properties": {
61                  "x": {
62                      "type": "number",
63                      "description": "The coordinate of the front left corner of the unrotated
                            resource in the data center layout width axis where the given resource is
                            located. Units are in millimeters.",
64                      "required": "true"
65                  },
66                  "y": {
67                      "type": "number",
68                      "description": "The coordinate of the front left corner of the unrotated
                            resource in the data center layout depth axis where the given resource is
                            located. Units are in millimeters.",
69                      "required": "true"
70                  },
71                  "rotation": {
72                      "type": "number",
73                      "description": "The rotation (degrees) from 0-359 around the center of the
                            resource.",
74                      "default": "0"
75                  },
76                  "resourceUri": {
77                      "type": "string",
78                      "description": "The uri of the rack resource whose position is described. "
79                  }
80              }
81          },
82          "description": "The collection of physical resources (racks) in the data center and their
                position."
83      },
84      "category": {...},
```

Note: In JSON, *integer* are integral numeric values (3, -24 …) while *number* can be an integer or a floating-point value (3, -24, 1.5, -3.3333 …)

We understand from this exploration of the datacenter schema that a datacenter is composed of collection of zero or more racks with their *resourceUri*.

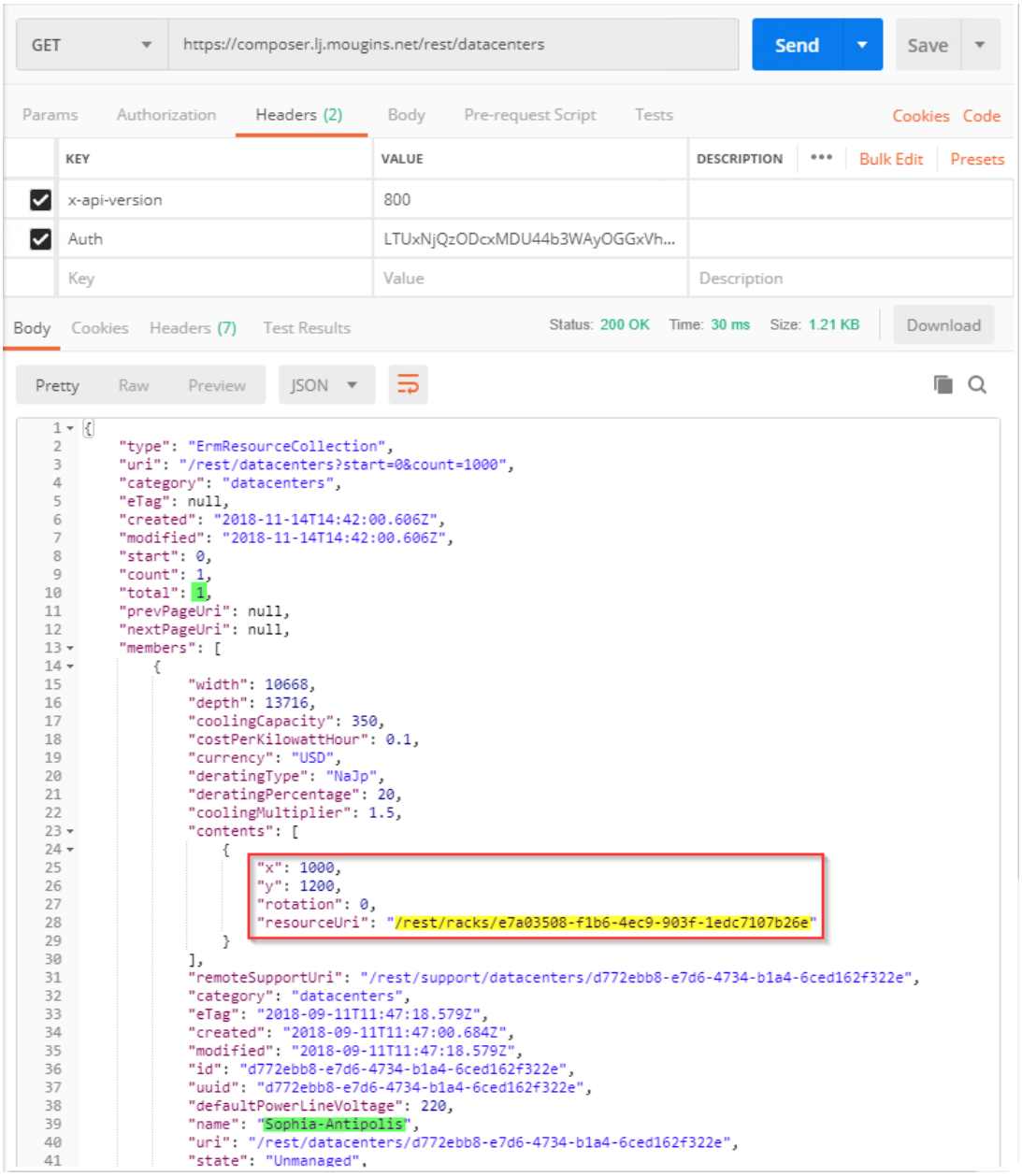
Read The Fantastic Manual, remember? In the *OneView API Reference* document the same object property information that we discover here is all listed in the Response body section.

If you do a search for **GET /rest/datacenters** and scroll down to **Response Body**, you will find the same properties and descriptions of each object available in the response body, useful if you need to find a specific property for your script developments:



So now, we know enough to start our first inventory and explore the topmost object instances found in our environment, the datacenters.

Let us remove the `/schema` from the Postman URL, and send the call using this time: https://composer.lj.lab/rest/datacenters

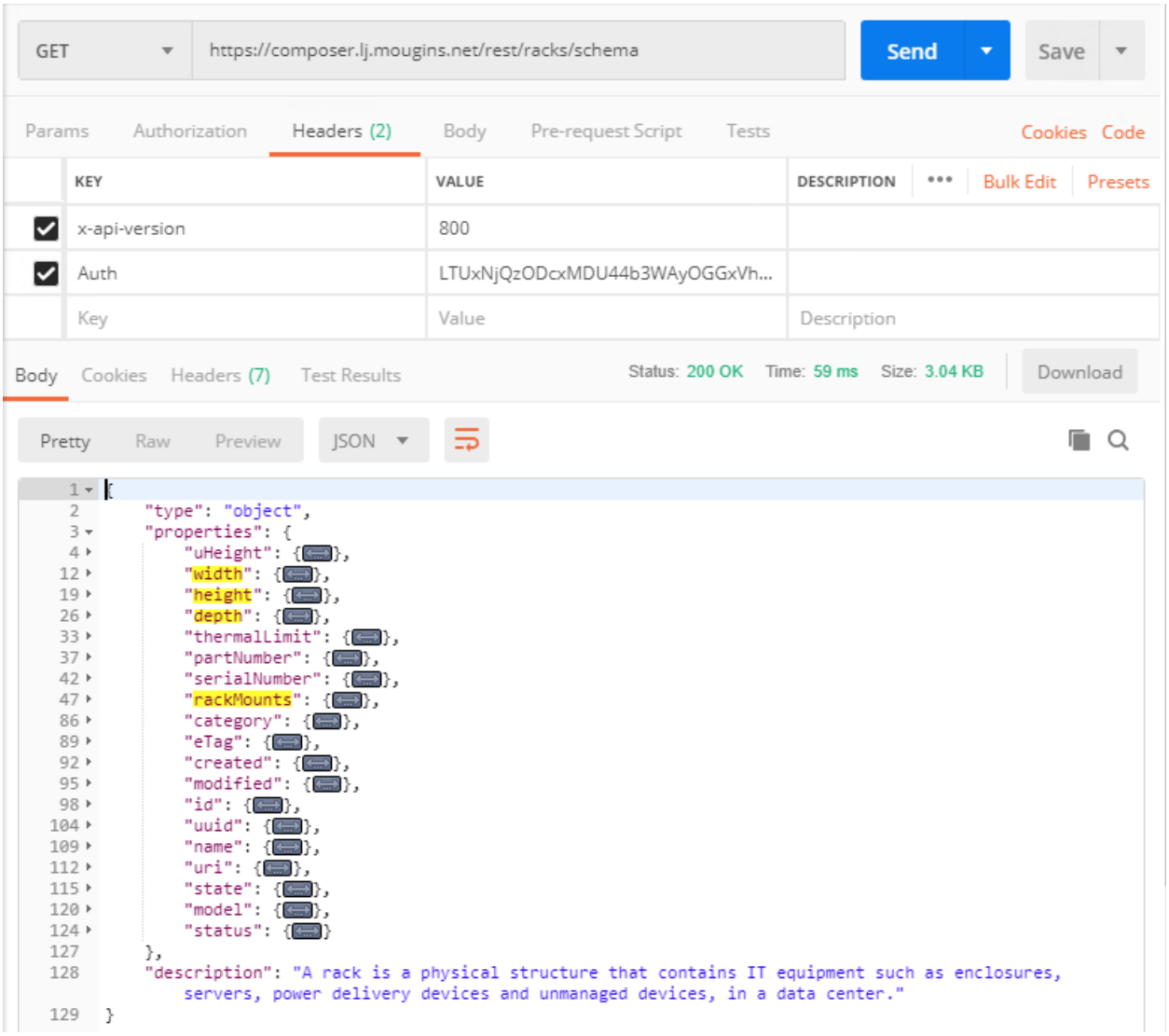


From the response JSON we can see that we get the same object information as with the `/schema` but this time with the actual datacenters' values.

We can discover that there is only one datacenter called "*Sophia-Antipolis*" and in this datacenter, there is a single rack, we have its position (x,y) and URI: `/rest/racks/e7a03508-f1b6-4ec9-903f-1edc7107b26e`

Note: REST APIs uses URI (Universal Resource Identifier) to identify/address resources

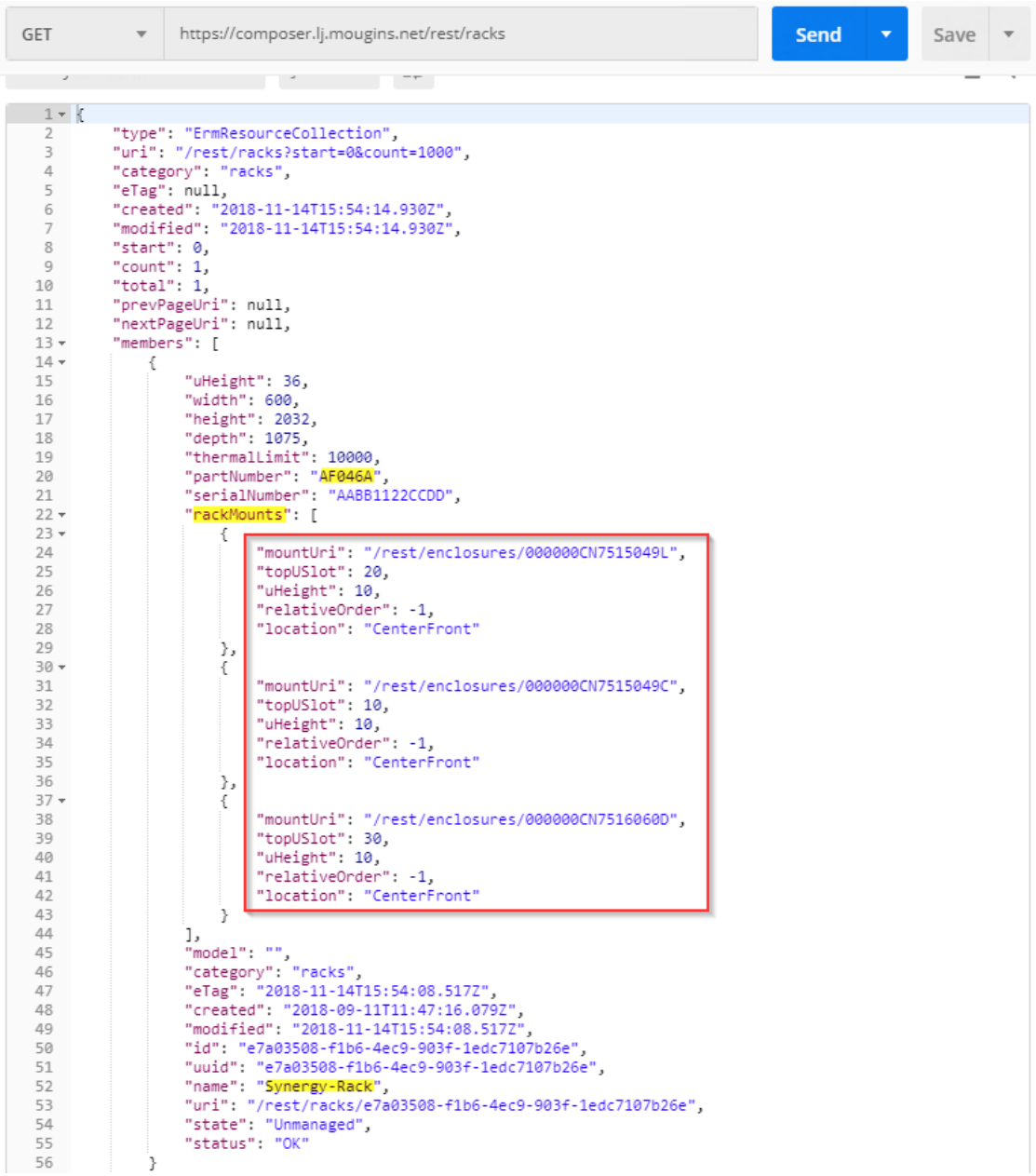
What is a Rack?
We can now drill down to the content of rack `e7a03508-f1b6-4ec9-903f-1edc7107b26e`
But before we start exploring the content of our rack, let us look at the schema for a rack by using the `/rest/racks/schema` URL using our REST client tool.



We can discover the properties of a rack, such as *width*, *height*, *depth*, and a collection of *rackMounts*, which represent zero or more items contained in the rack (*enclosure*, *server*-hardware, *power delivery device*)

We can expand the *rackMounts* property to understand what composes a rack. We discover that it is made of a collection of items (*array*), and that each item has a *mountUri* to access it.

Let us now do a `GET` request on `/rest/racks`.



From that query, we discover that our rack is named "*Synergy-Rack*", its part number is *AF046A* and in this rack we have 3 x 10U enclosures. We have, with the *mountUri* property, a direct access to the URI of each enclosure in the rack.

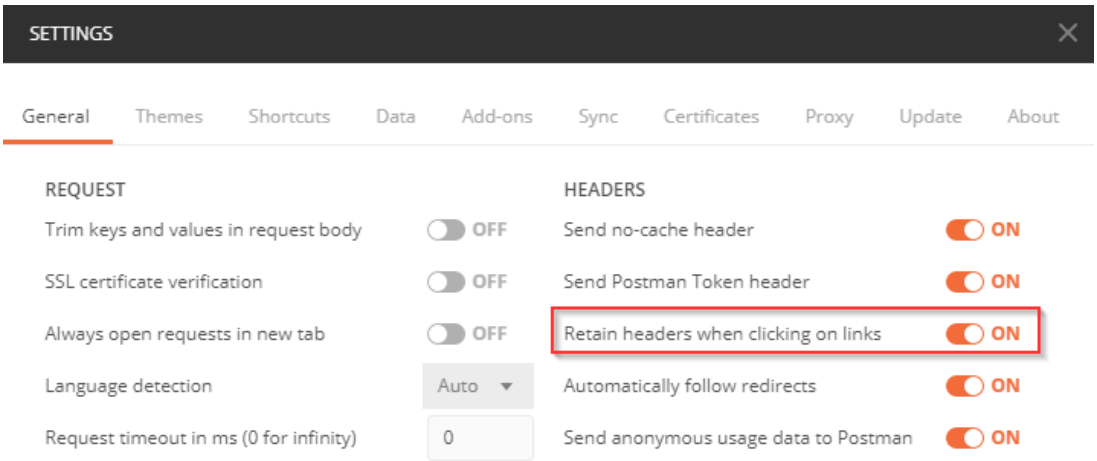
What is an Enclosure?
We can go further down by taking a few minutes to explore an enclosure.

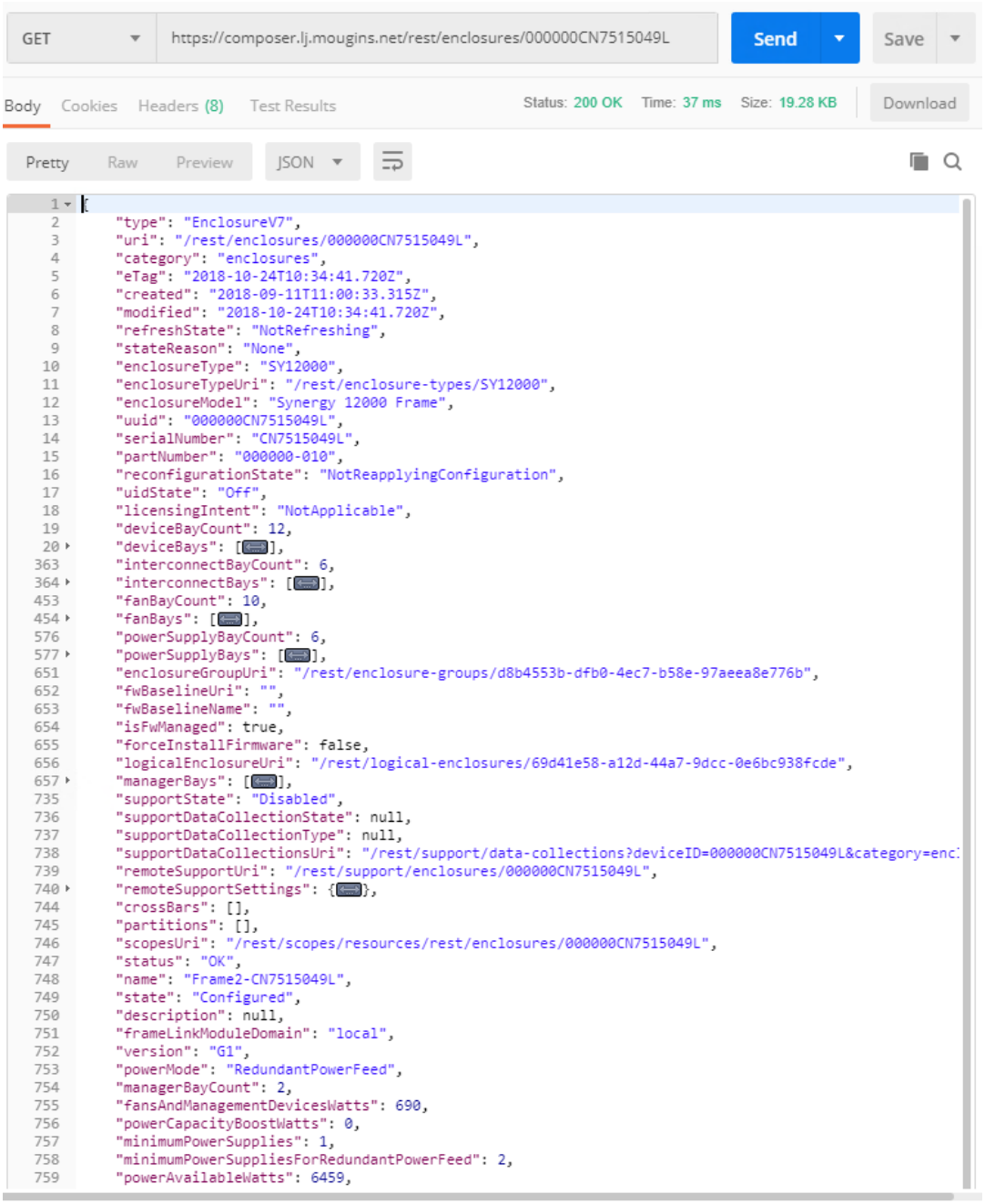
One of the cool Postman features is that you can click on links inside the response body, so click on one of the enclosures *mountUri* link:


```
"members": [
    {
        "uHeight": 36,
        "width": 600,
        "height": 2032,
        "depth": 1075,
        "thermalLimit": 10000,
        "partNumber": "AF046A",
        "serialNumber": "AABB1122CCDD",
        "rackMounts": [
            {
                "mountUri": "/rest/enclosures/000000CN7515049L",
                "topUSlot": 20,
                "uHeight": 10,
                "relativeOrder": -1,
                "location": "CenterFront"
            },
```


This will automatically load a `GET` Request in Postman with the link URL and retains our very important headers content (auth and x-api-version) then click **Send**

Note: If you don't get the header content retained, make sure the option is enabled in Postman settings:

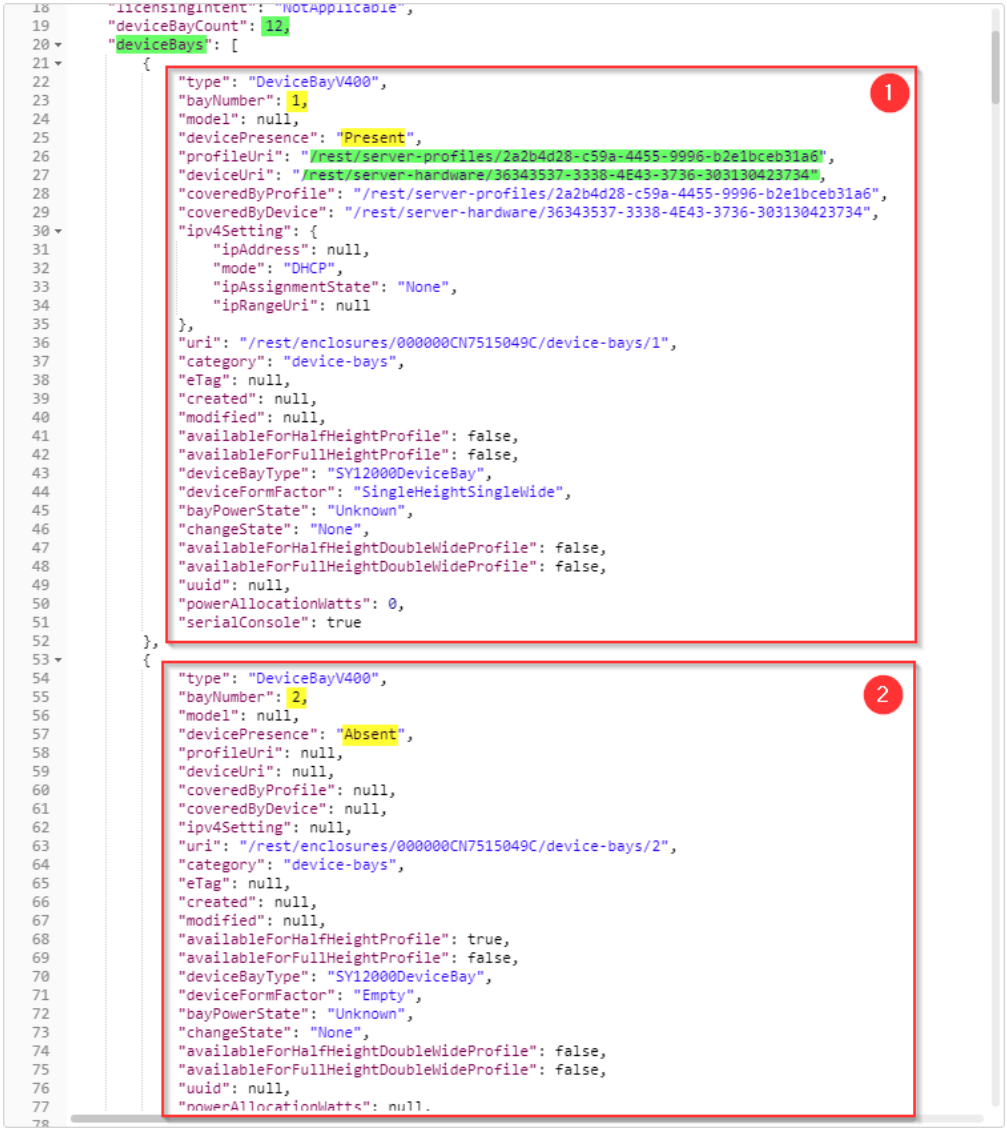




We discover many elements and useful parameter that make up an enclosure, such as name, Serial Number (*serialNumber*), Enclosure Type (*enclosureType*), Composable Infrastructure Appliances (*managerBays*), Interconnect Modules (*interconnectBays*), Power Supplies (*powerSupplyBays*), Compute Modules (*deviceBays*), etc.

We can also see that an enclosure is composed of a *deviceBays* array of items.

We can also discover that it is composed of 12 device bays. Each item in *devicebays* contains the description of a device bay. We see the bay number, the presence of a server hardware, if a profile is assigned, etc.




```
"licensingIntent": "NotApplicable",
"deviceBayCount": 12,
"deviceBays": [
    {
        "type": "DeviceBayV400",
        "bayNumber": 1,
        "model": null,
        "devicePresence": "Present",
        "profileUri": "/rest/server-profiles/2a2b4d28-c59a-4455-9996-b2e1bceb31a6",
        "deviceUri": "/rest/server-hardware/36343537-3338-4E43-3736-303130423734",
        "coveredByProfile": "/rest/server-profiles/2a2b4d28-c59a-4455-9996-b2e1bceb31a6",
        "coveredByDevice": "/rest/server-hardware/36343537-3338-4E43-3736-303130423734",
        "ipv4Setting": {
            "ipAddress": null,
            "mode": "DHCP",
            "ipAssignmentState": "None",
            "ipRangeUri": null
        },
        "uri": "/rest/enclosures/000000CN7515049C/device-bays/1",
        "category": "device-bays",
        "eTag": null,
        "created": null,
        "modified": null,
        "availableForHalfHeightProfile": false,
        "availableForFullHeightProfile": false,
        "deviceBayType": "SY12000DeviceBay",
        "deviceFormFactor": "SingleHeightSingleWide",
        "bayPowerState": "Unknown",
        "changeState": "None",
        "availableForHalfHeightDoubleWideProfile": false,
        "availableForFullHeightDoubleWideProfile": false,
        "uuid": null,
        "powerAllocationWatts": 0,
        "serialConsole": true
    },
    {
        "type": "DeviceBayV400",
        "bayNumber": 2,
        "model": null,
        "devicePresence": "Absent",
        "profileUri": null,
        "deviceUri": null,
        "coveredByProfile": null,
        "coveredByDevice": null,
        "ipv4Setting": null,
        "uri": "/rest/enclosures/000000CN7515049C/device-bays/2",
        "category": "device-bays",
        "eTag": null,
        "created": null,
        "modified": null,
        "availableForHalfHeightProfile": true,
        "availableForFullHeightProfile": false,
        "deviceBayType": "SY12000DeviceBay",
        "deviceFormFactor": "Empty",
        "bayPowerState": "Unknown",
        "changeState": "None",
        "availableForHalfHeightDoubleWideProfile": false,
        "availableForFullHeightDoubleWideProfile": false,
        "uuid": null,
        "powerAllocationWatts": null,
```

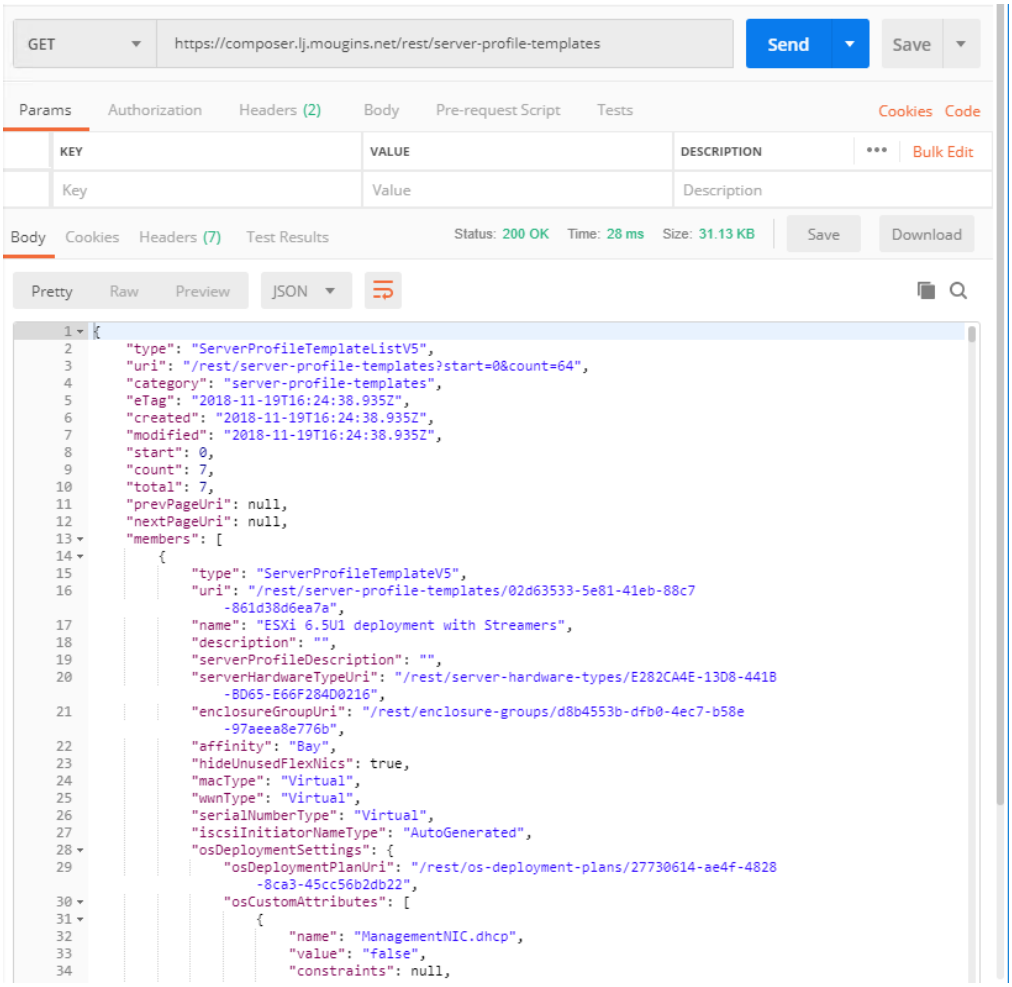

In our case, bay 1 is populated with a Synergy Compute module for which we are given the server-hardware resource, *deviceUri* (`/rest/server-hardware/36343537-3338-4E43-3736-303130423734`) and the profile resource *profileUri* currently assigned to that bay (`/rest/server-profiles/2a2b4d28-c59a-4455-9996-b2e1bceb31a6`). On the other hand, Bay 2 is empty and without any profile assigned.

We now understand that an HPE OneView instance contains a collection of one or more datacenters, which is composed of a collection of zero or more racks, which contains a collection of one or more items such as a server, enclosure, profiles, etc.

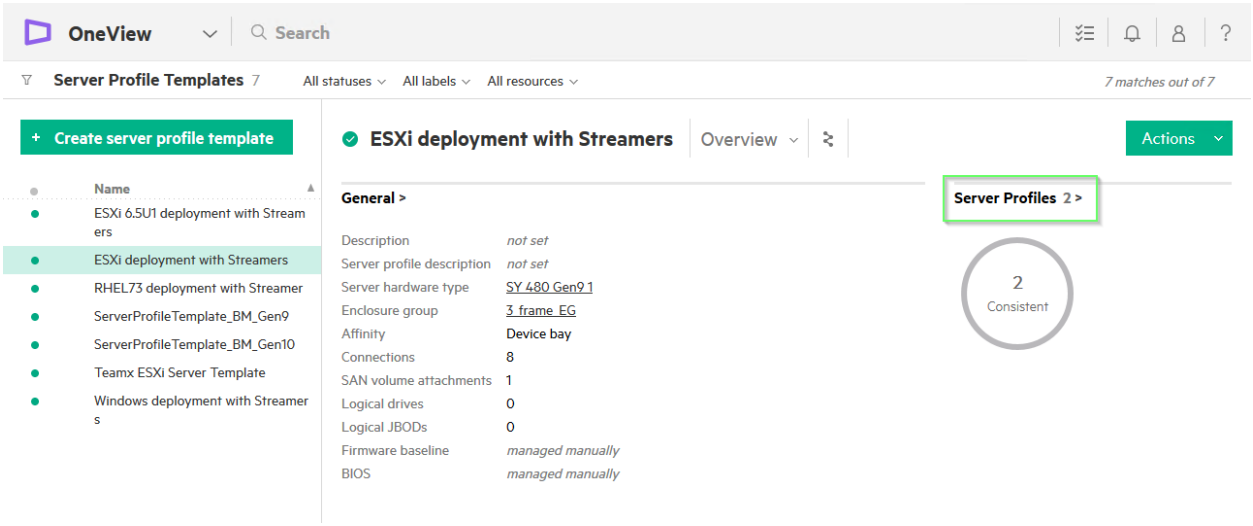
Associations

We just discovered the relationships between the HPE OneView resources, this is an important subject. Very often when retrieving information from the API, it can be difficult to figure out which child resources are in association with the current resource.

If you send a `GET /rest/server-profile-templates`, you will see that the response body does not provide any information about the server profiles that are associated with that Server Profile Template.

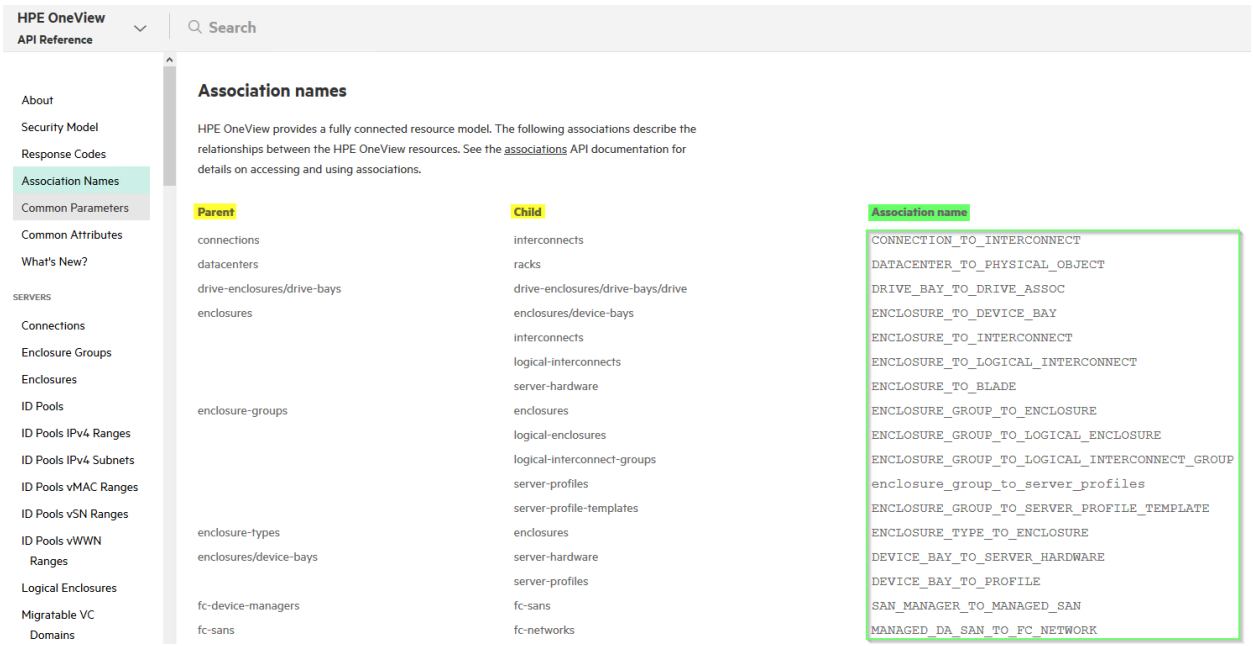


This may seem particularly unexpected when you know that this information is directly available in the Overview pane:

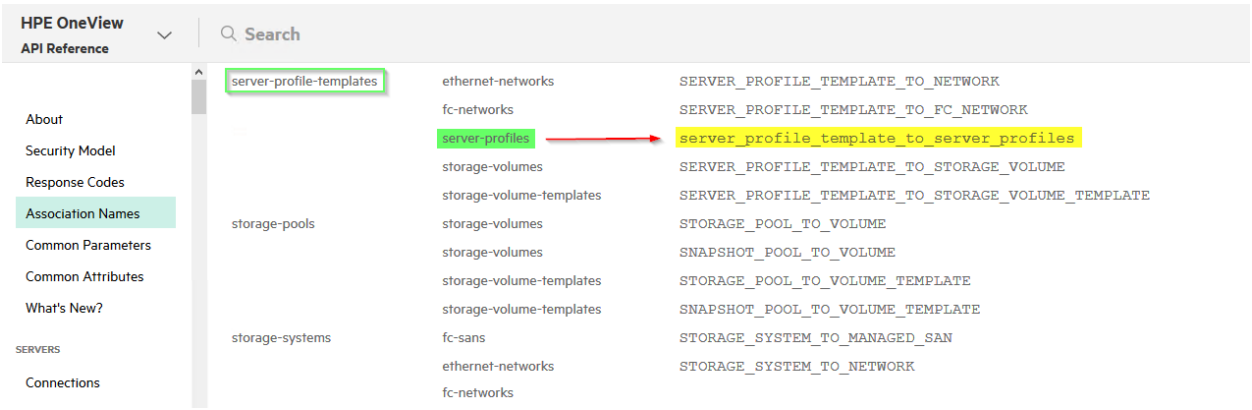


Additionally, enquiring a list of server profiles managed by a server profile template is a quite a basic request and you might struggle a long time to find out this information using a simple GET call on the resource itself…

The good news is that there is an easy way to find that, using an API feature called **Associations names**. OneView provides a fully connected resource model with **child/parent associations** and these associations are described in the API reference documentation:

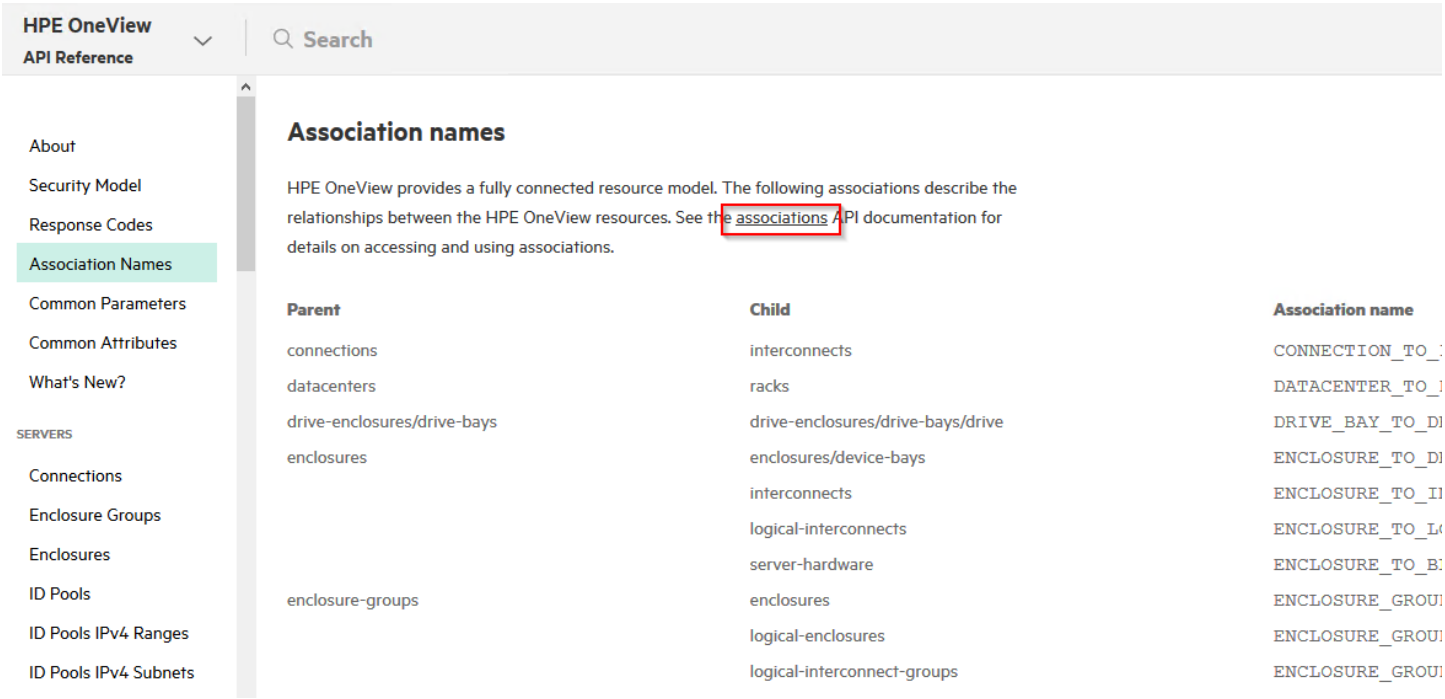


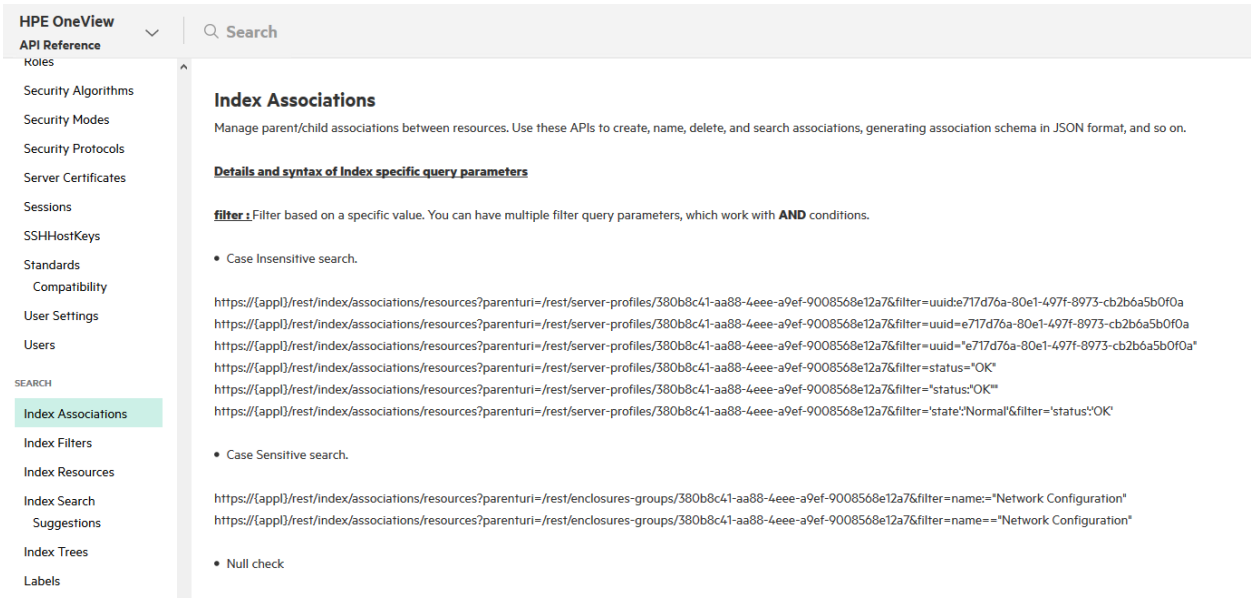
Back to our Server Profile Template issue, if you scroll down to the **Server-Profile-Templates** parent, there is a **Server-Profile** child item with its associated name: *server_profile_template_to_server_profiles*



This association name *server_profile_template_to_server_profiles* can be used as a query name.
name=server_profile_template_to_server_profiles

Scroll back to the begin of the **Association Names** web page and click on **associations**.





In this page, we can find several examples of API request to create, delete, name and search associations. The query parameters to get the list of associations that match the requested parameters use the following syntax: `https://{appl}/rest/ index/associations?parenturi={resource URI}&{parameters}`

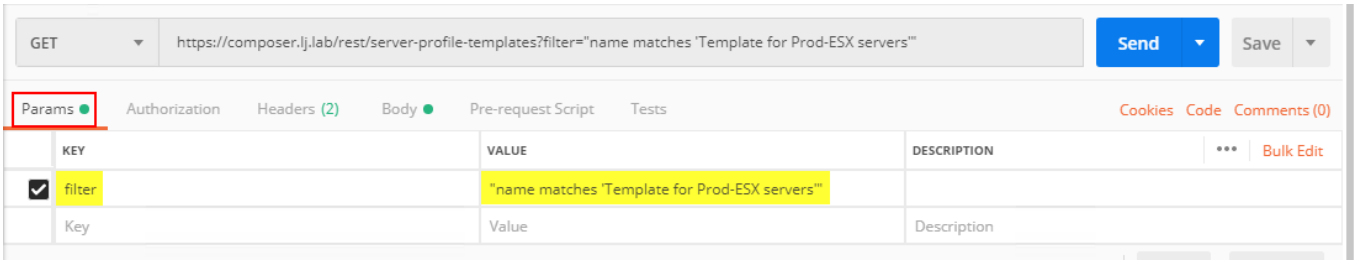
The query parameters can be used with `parenturi`/`childuri`/`name` filtering.

As an example, use Postman with the following parameters to get first the URI of an existing server profile template:

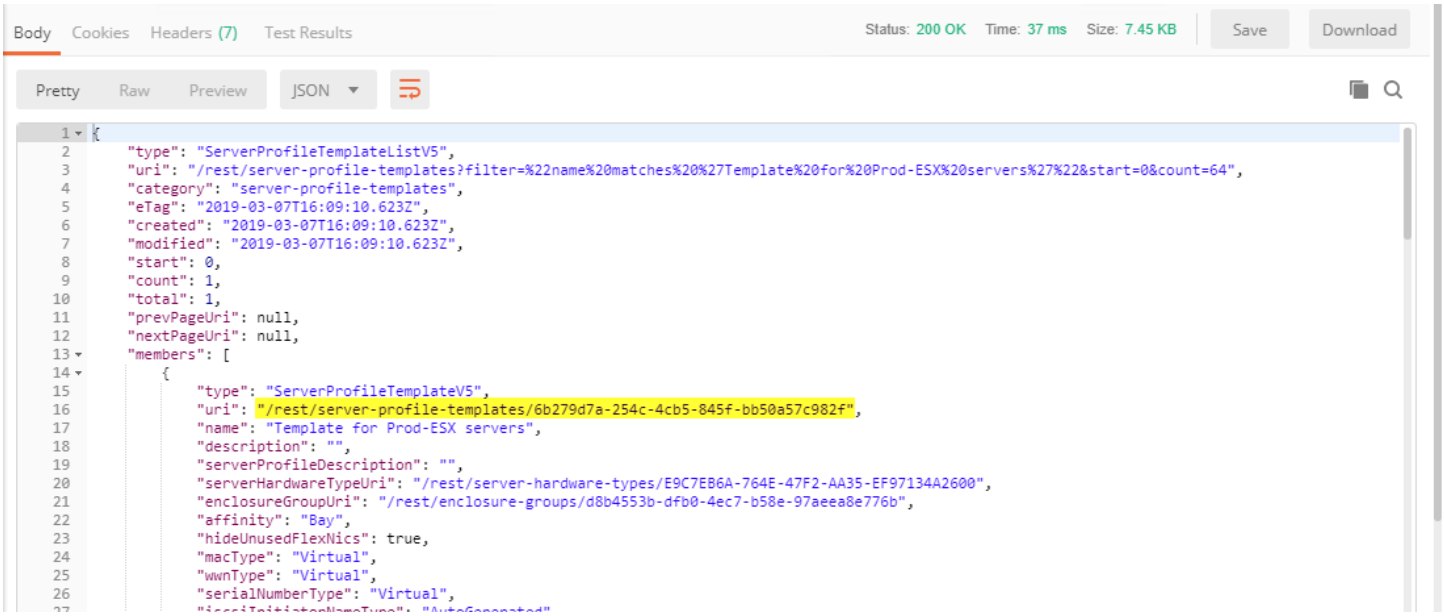
Field	**Value**
URL	`https://composer.lj.lab/rest/server-profile-templates`
Params Key	`filter`
Params Value	`"name matches 'Template for Prod-ESX servers'"`
Verb	`GET`
Header	`Auth : your sesionID` `X-API-Version:800`

This request is a little bit different than what we saw previously as we are using here a query parameter to filter the response to only get the Server Profile Template that matches with the name “Template for Prod-ESX servers”.

Notice that using those parameters modify the URL to `/rest/server-profile-templates?filter="name matches 'Template for Prod-ESX servers'"`



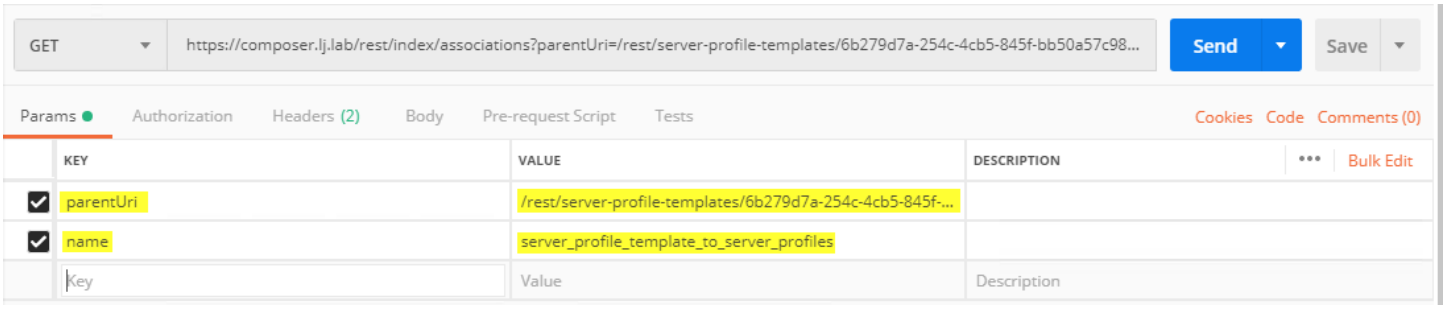
The response is then filter with only the Server Profile Template we are looking for.



Now record the URI of the Server Profile Template highlighted above as we are going to use it in the next request.

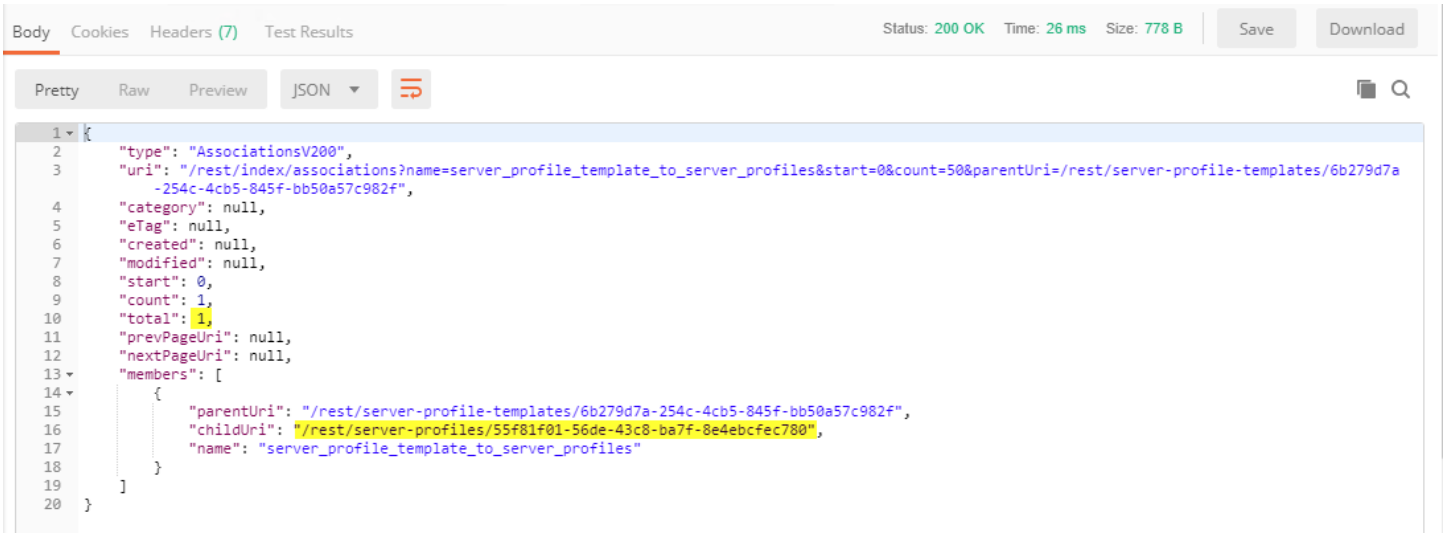
Now let's create a new Postman request to get the association between a Server Profile Template and Server Profiles using the association name `server_profile_template_to_server_profiles` and the parentURI of our Server Profile Template recorded previously:

Field	Value
URL	`https://composer.lj.lab/rest/index/associations`
Parameter key	`parentUri`
Parameter value	`/rest/server-profile-templates/6b279d7a-254c-4cb5-845f-bb50a57c982f`
Parameter key	`name`
Parameter value	`server_profile_template_to_server_profiles`
Verb	`GET`
Header	`Auth : your sesionID` `X-API-Version:800`



Notice that using those parameters modify the URL to `https://composer.lj.lab/rest/index/associations?parentUri=/rest/server-profile-templates/6b279d7a-254c-4cb5-845f-bb50a57c982f&name=server_profile_template_to_server_profiles`

This is what the response body looks like:



The response shows that the Server Profile Template `/rest/server-profile-templates/6b279d7a-254c-4cb5-845f-bb50a57c982f` has one Server profile `/rest/server-profiles/55f81f01-56de-43c8-ba7f-8e4ebcfec780` associated with it.

You can click on the *childUri* link and click **Send** to access to the server profile information:

This was an example of how to use associations to find which child resources are in association with a parent resource. Other association requests using filtering (created, modified, etc.), conditions (and, or, not) and regular expression can also be used.

This concludes Lab-1 to 5 and the OneView API programing.

You know enough now to run your own invocation. We showed you how API invocation works, how to get authenticated, how to find the information and how to create your own API call.

For fun, you can try to solve the following exercises to validate your skills:

Exercise 1
How can you get using a REST call the number of Ethernet network managed by OneView that have smartlink enabled and with an iSCSI purpose?
Hint: RTFM, all can be found in the documentation!
Answer: Look at the bottom of this page.

Exercise 2
How can you configure a remote SYSLOG server in the HPE OneView GUI?
Hint: Take a look at the REST API documentation
https://composer.lj.lab/api-docs/current/#rest/remote-syslog
This example illustrates the need to have a good understanding of the REST API because SYSLOG in OneView 4.1 is one of the features that can only be enabled through the RESTful API.

In the next Lab, we will discover the HPE OneView Global Dashboard API and how we can interact with it.

Answer to exercise 1:
GET https://composer.lj.lab/rest/ethernet-networks?filter=smartLink=true&filter=purpose=ISCSI

Lab 6: HPE Global Dashboard API

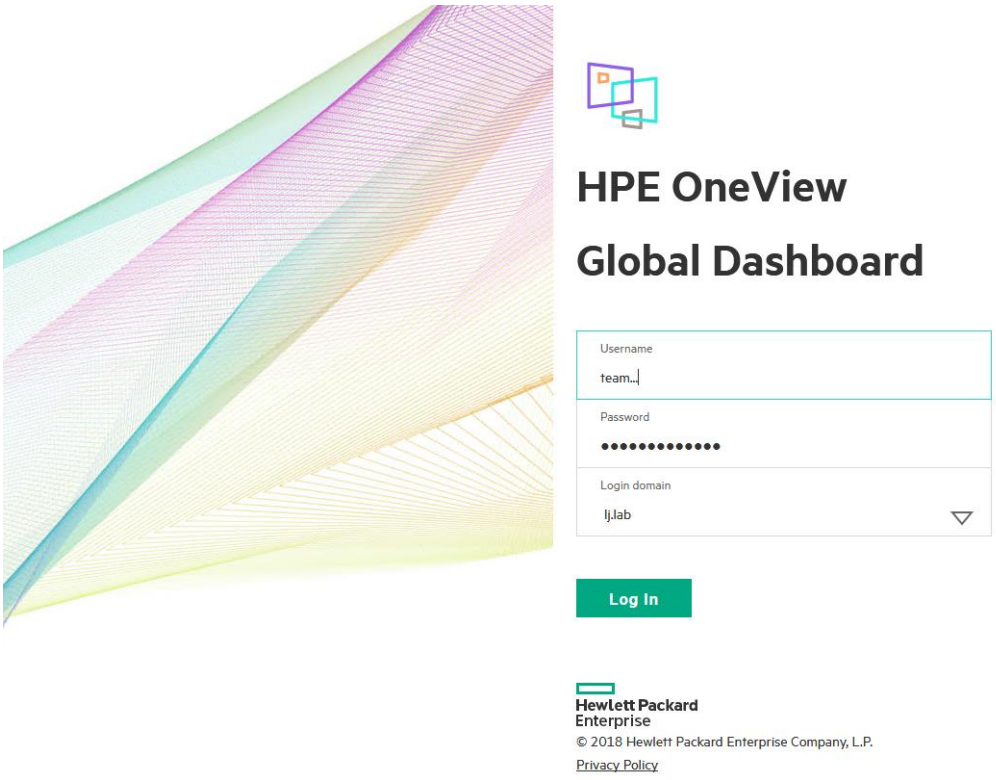
HPE OneView Global Dashboard is a standalone appliance which provides a simple, efficient, and unified view of the health of servers, profiles, and enclosures across multiple HPE OneView appliances, HPE Synergy, HPE 3PAR and SimpliVity systems.

With HPE OneView Global Dashboard 1.6, released in October 2018, a RESTful API is available for the first time. A RESTful API means that you can query all the OneView appliances that have been added to Global Dashboard via command line, and even scripts if you are ambitious enough.

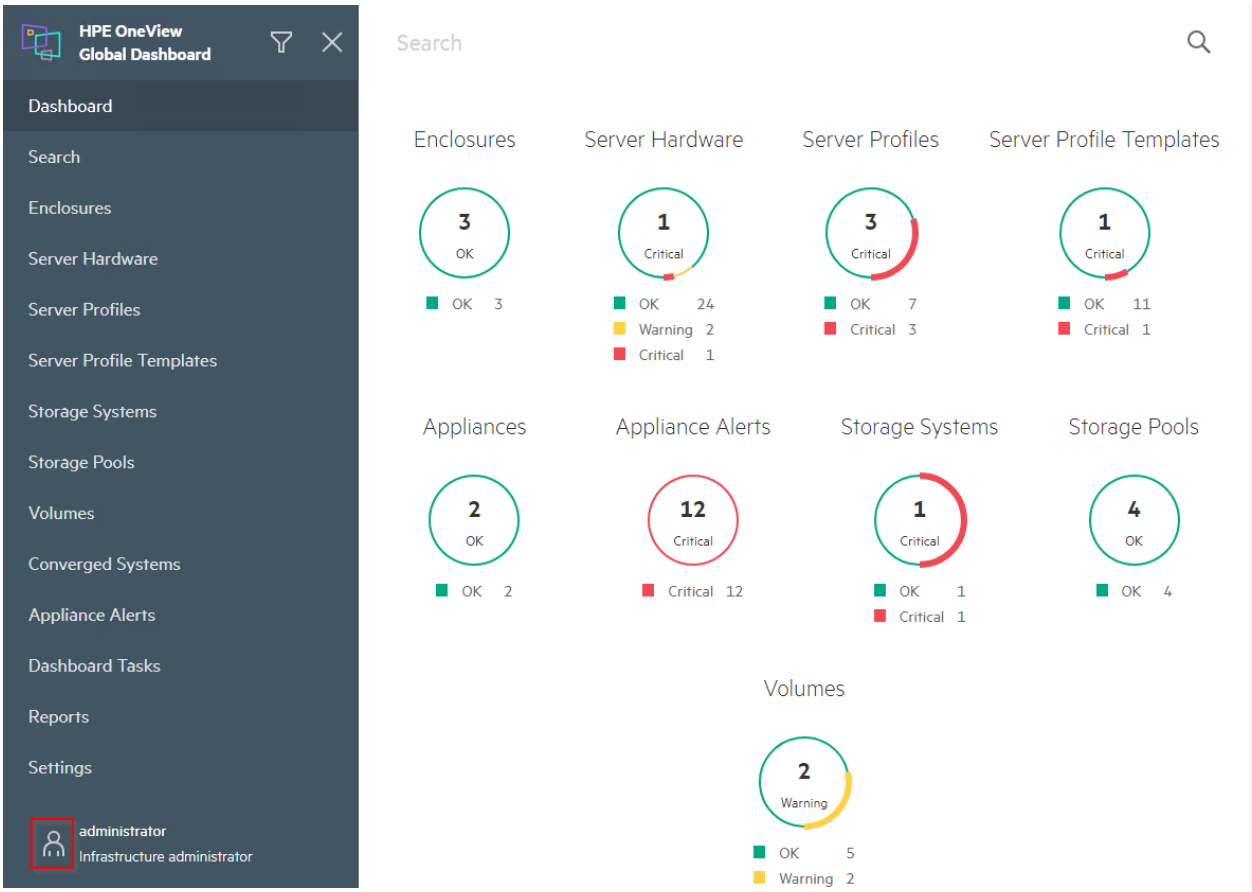
Login to the HPE OneView Global Dashboard (https://oneview-global-dashboard.lj.lab) using the following credentials:

- Username: **teamx**
- Password: **Passwordteamx**
- Domain: **lj.lab**

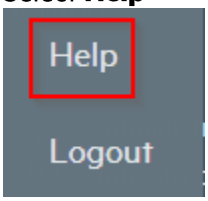
with **x** being your team number



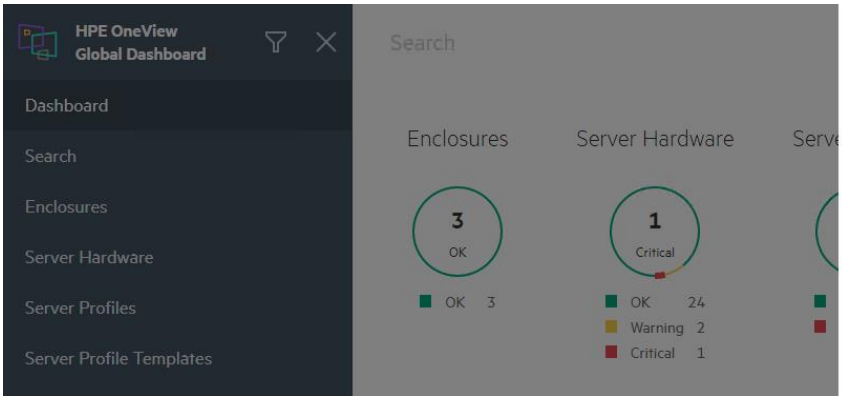
Once connected, click the user icon to open the session menu, which provides options to access online help, REST API reference, and logout.

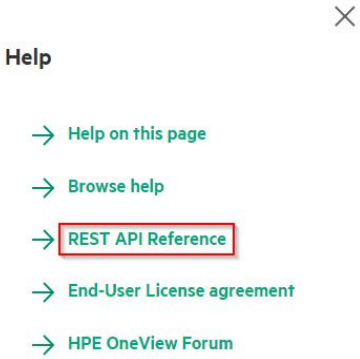


Select **Help**



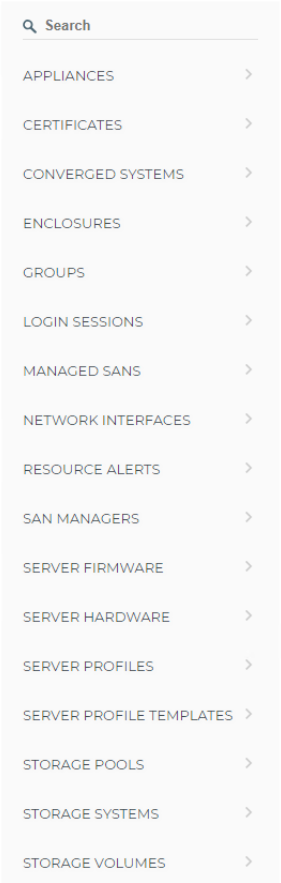
Then click on **REST API Reference**

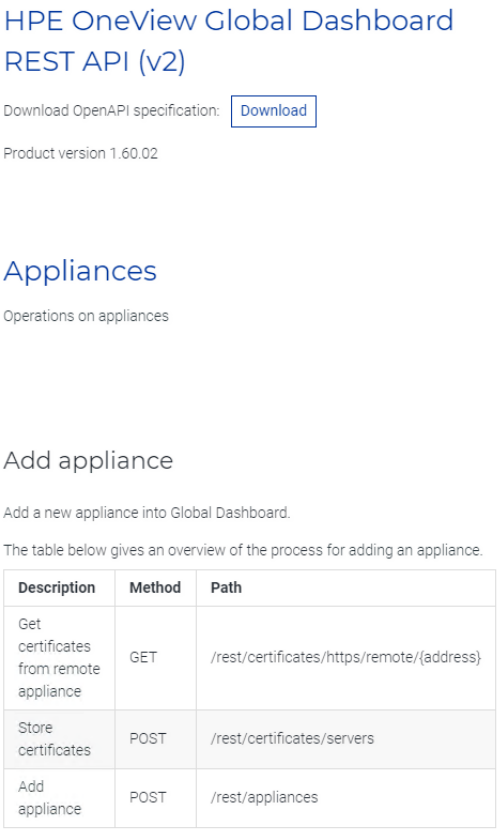


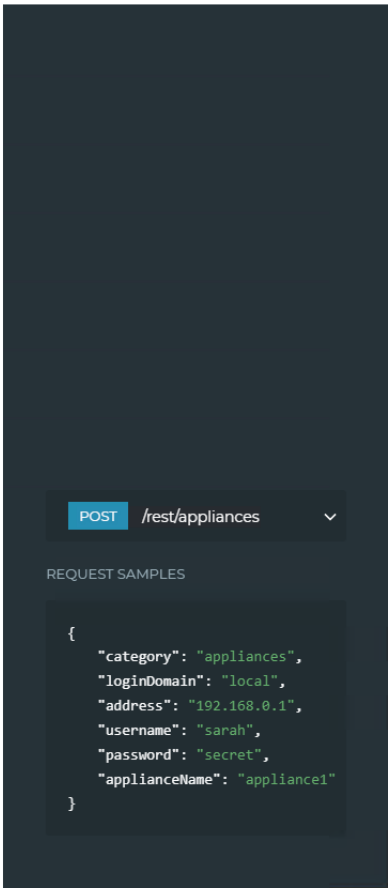


This opens the REST API reference in a new tab.

Like under HPE OneView, the API reference document provides all necessary information about the RESTful API.

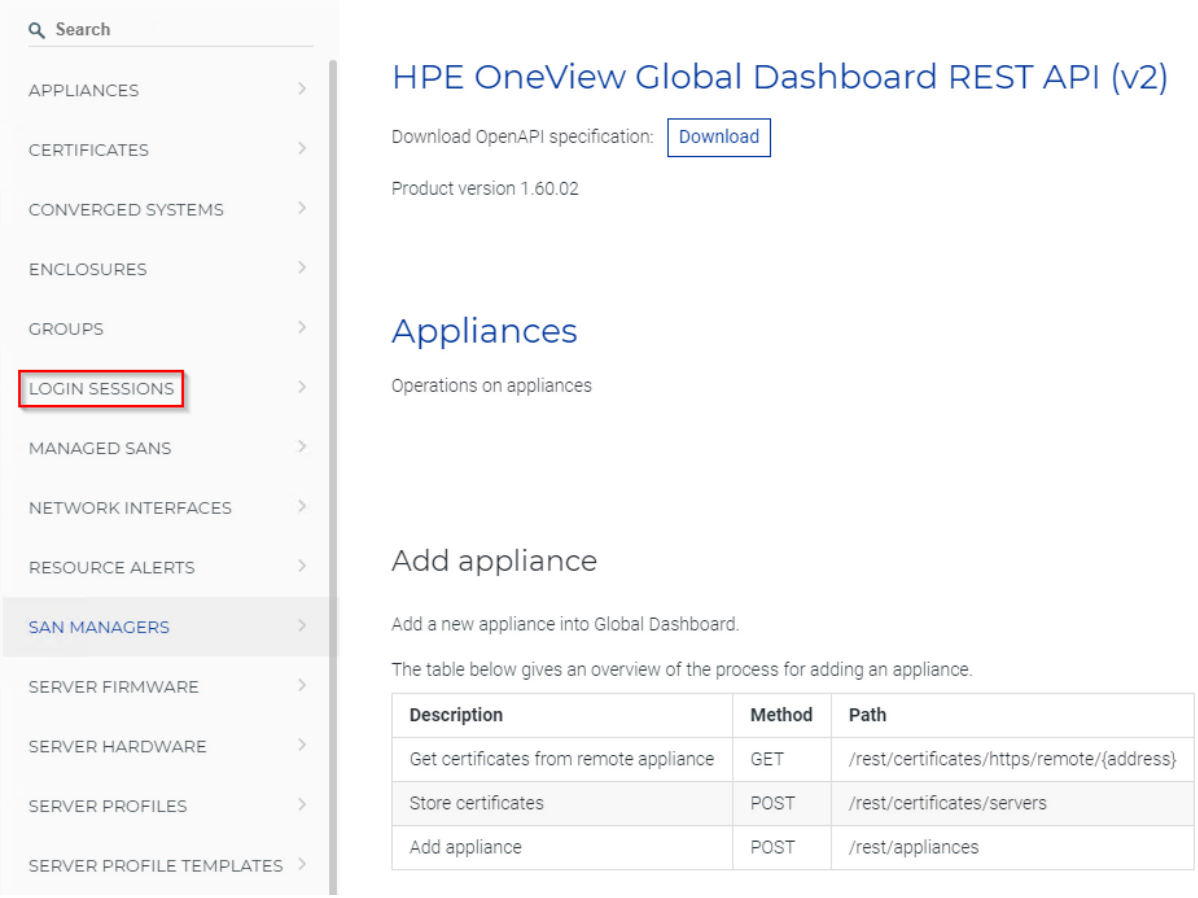






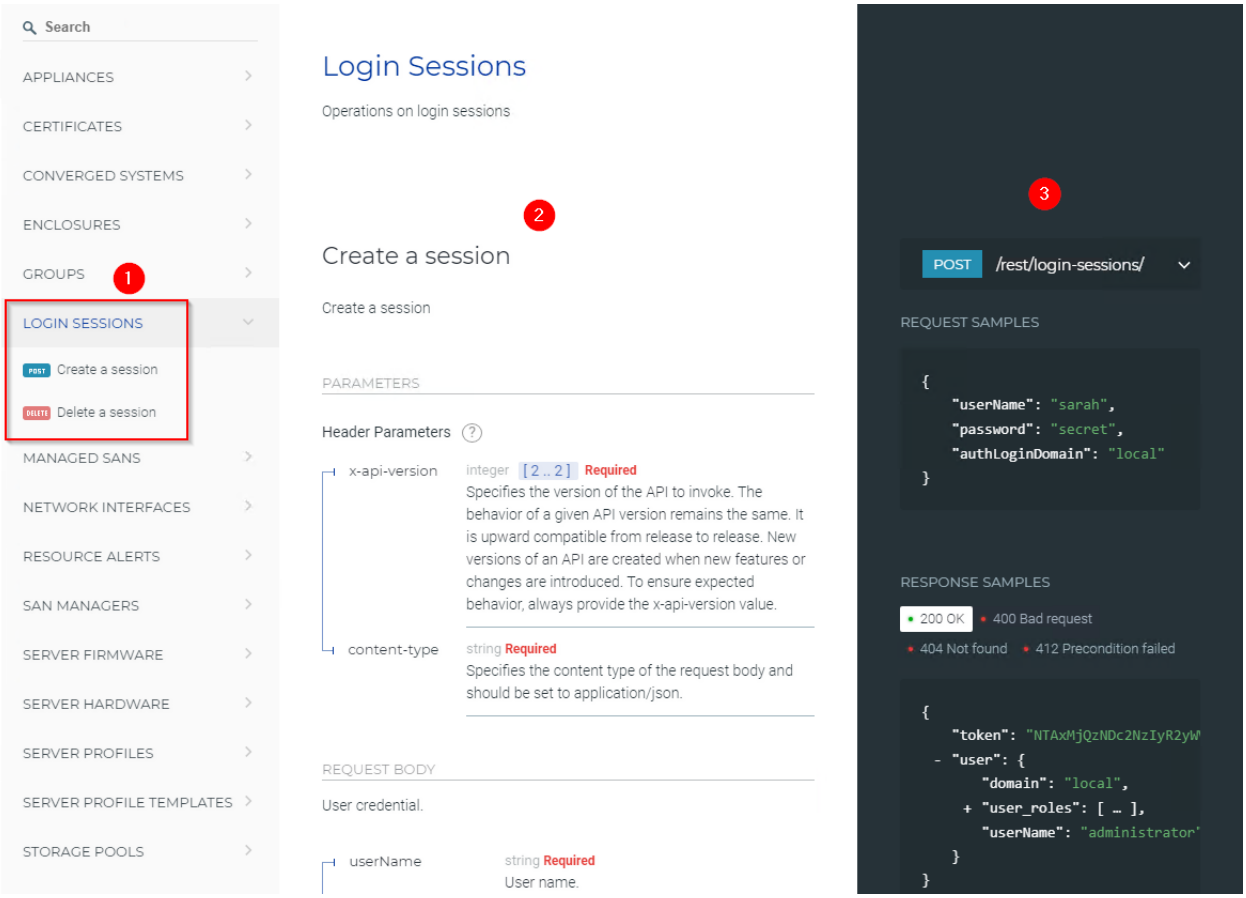
On the left pane, you can find the list of resources supported by the API.

Click on **Login Sessions**

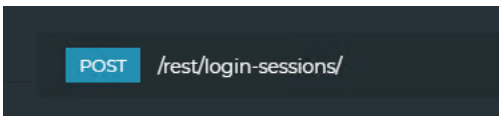


For each resource, the guide provides the following information:

1- A list of supported API request (`GET`, `POST`, `DELETE`, etc.)
2- A description of the specific task performed by each API request with a list of all required parameters for the header and body.
3- A pane with samples for API request and response.

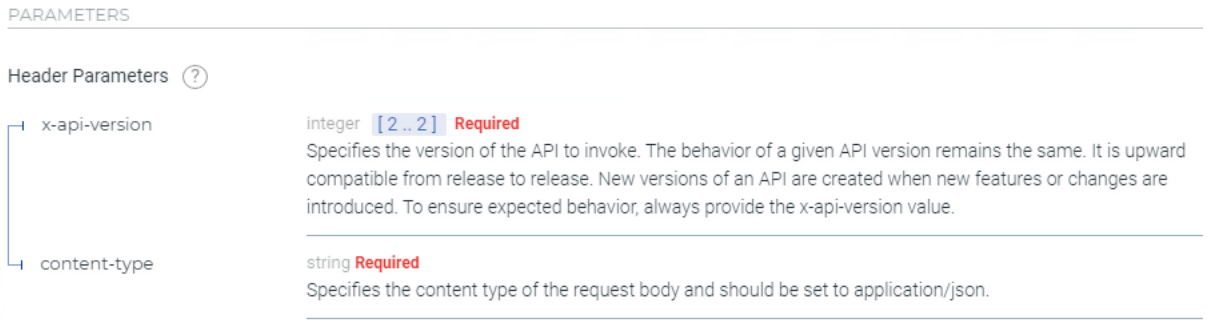


For our first interaction with the Global Dashboard API, let's open a login session to HPE OneView Global Dashboard. From the documentation, we can see that in order to open (i.e. create) a login session, we need to use the `POST` method against the URI `/rest/login-sessions`.



We can also see that we need to set two HTTP Headers:

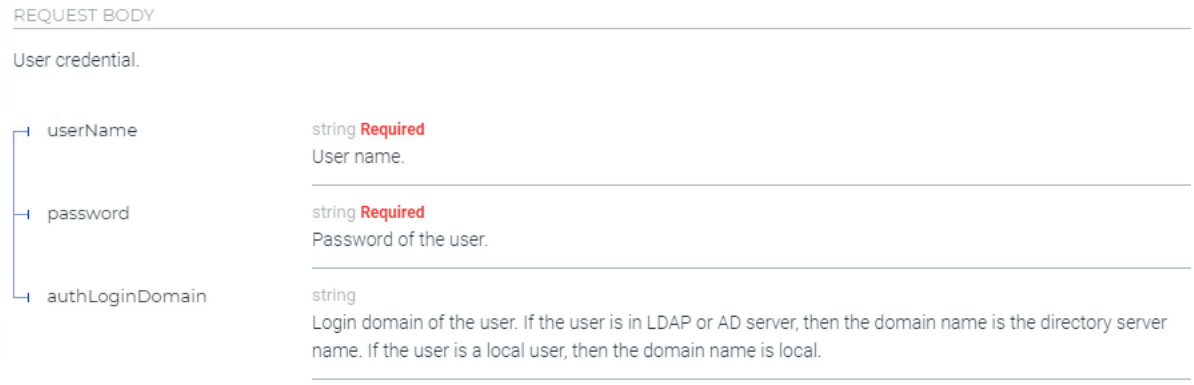
- **X-API-Version**: set to the version of the API we want to use (set to *2*)
- **Content-Type**: set to *application/json*, to tell the API, that the payload we will be providing is in JSON notation



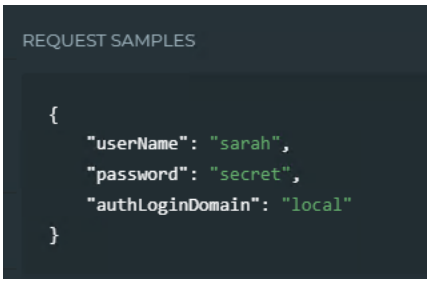
The payload expected by the API must contains (at minimum) the following parameters:

- **userName**: set to the user name to use for authentication
- **password**: set to the password for that user name
- **authLoginDomain**: set the login domain of the user

Note: Be careful, JSON is case sensitive. userName is expected here, not username, nor UserName. As a rule, the API expects lower case words, but capitalized, after the first word, if there are more than one word.



The payload is notified by the `{ … }` in the sample pane:



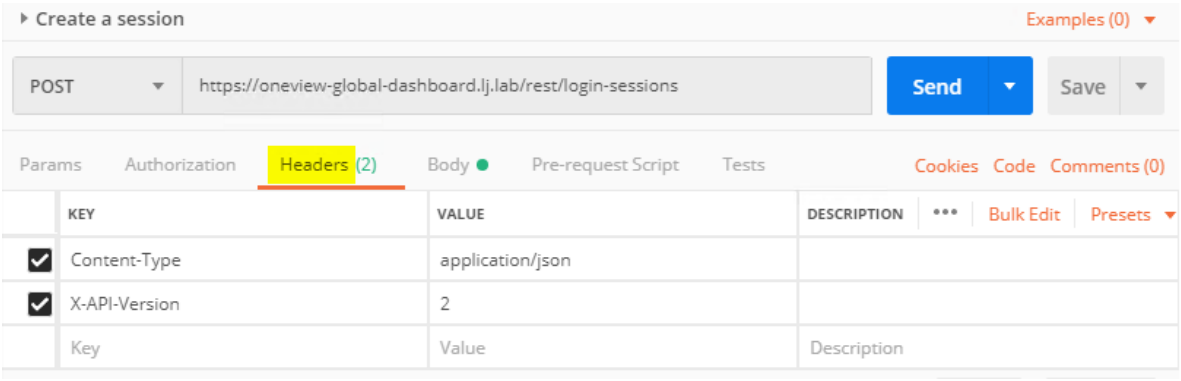
You can test this now using Postman, using the following parameters:

Field	Value
URL	`https://oneview-global-dashboard.lj.lab/rest/login-sessions`
Verb	`POST`
Header	`Content-type:application/json` `X-API-Version:2`

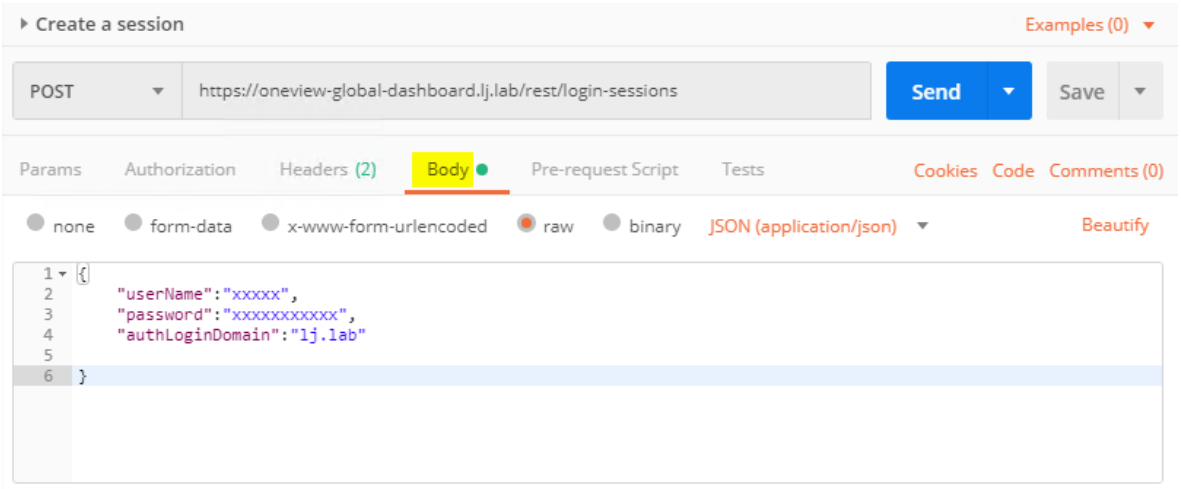
Payload	`{`
	`"userName": "xxxxx",`
	`"password": "xxxx",`
	`"authLoginDomain": "lj.lab"`
	`}`

Make sure you set your appropriate credentials for `userName` and `password`.

Content of the header:

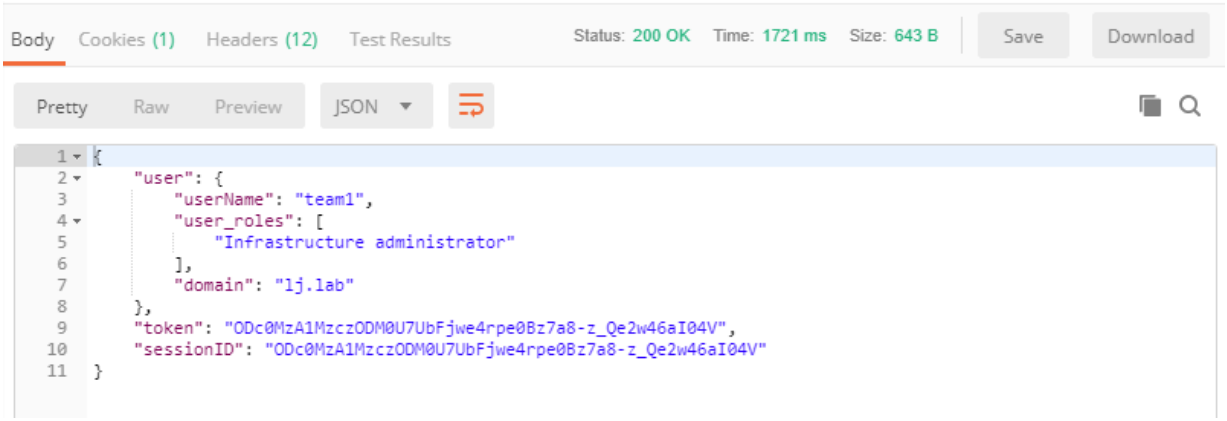


Content of the body payload:



Press **Send**.

The response we get is the following:

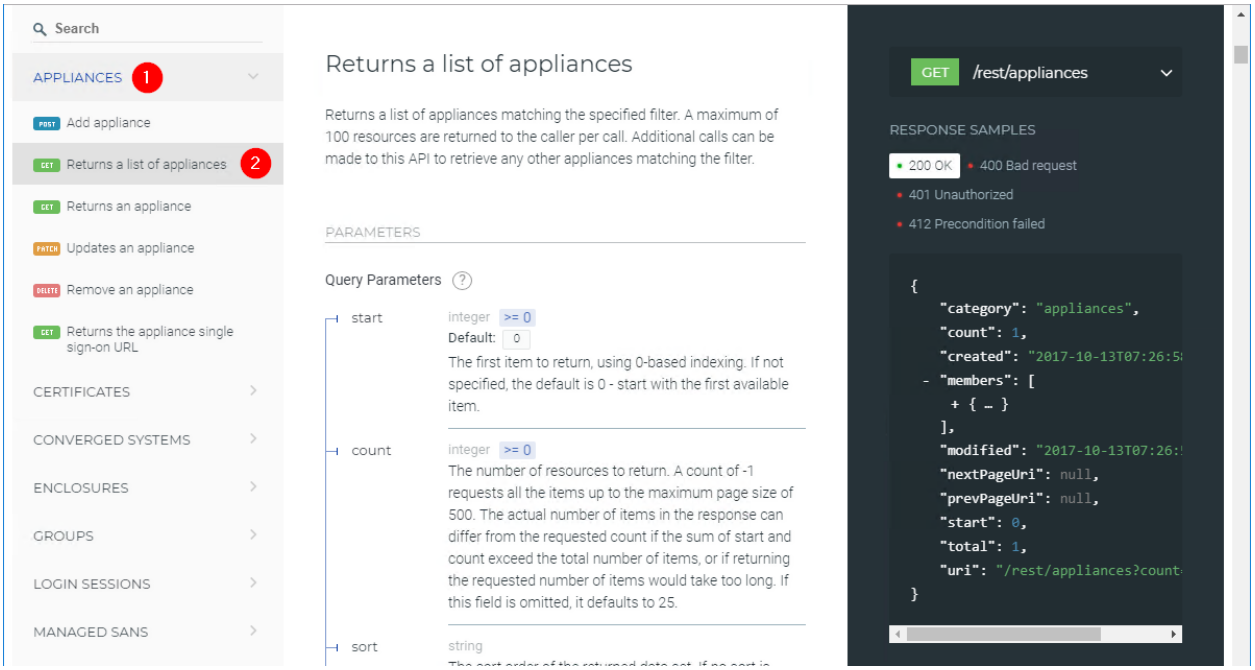


The *sessionID*, also known as the session token is what we are looking for. This is the token that will be used for authentication in the subsequent calls to the API.

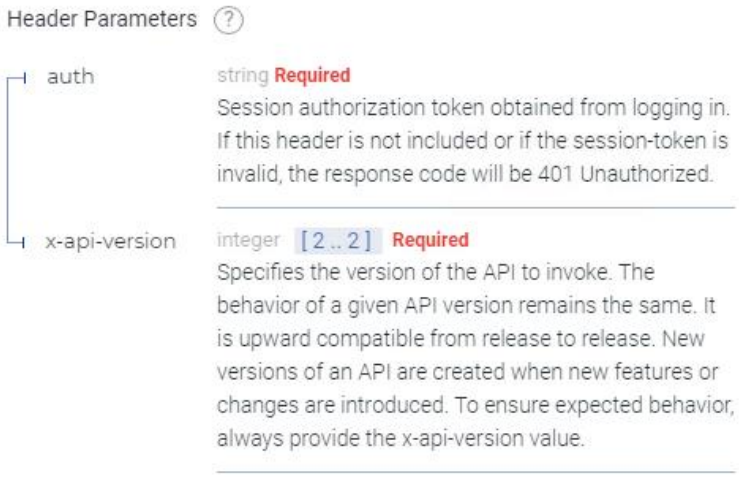
So, in order to be authorized to call any other API methods, we will need to provide the session token as another HTTP Header called `Auth`.

We can now test this token and call another API method. Let us pick `GET /rest/appliances` which return a list of appliances managed by Global Dashboard.

This is what we can read:



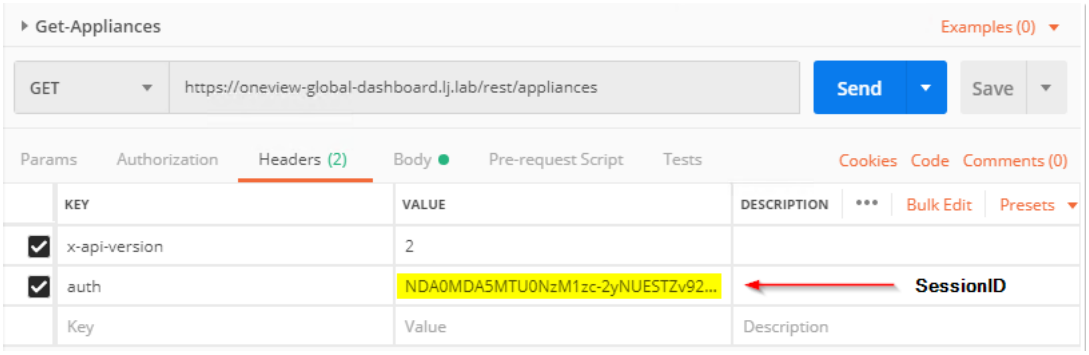
Notice now the presence of Auth with the sessionID in the header for this call.



So, use Postman with the parameters found here:

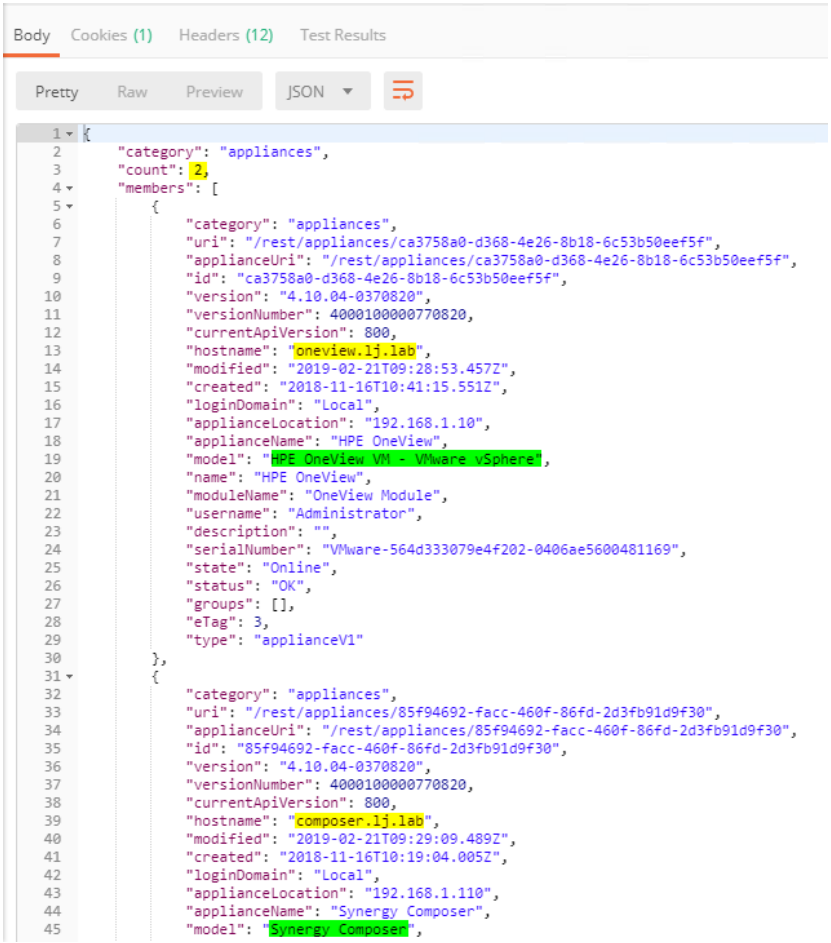
Field	**Value**
URL	`https://oneview-global-dashboard.lj.lab/rest/appliances`
Verb	`GET`
Header	`Auth : your sesionID` `X-API-Version:2`

So, it is necessary to copy/paste your *SessionID* from your `POST /rest/login-sessions` in the `auth` value field:



Then when all set, press **Send**

You should get the list of appliances managed by Global Dashboard in the response body, again in a JSON format. We can easily see that we have 2 appliances, one is a OneView VM VMware appliance, another is a OneView Synergy Composer:



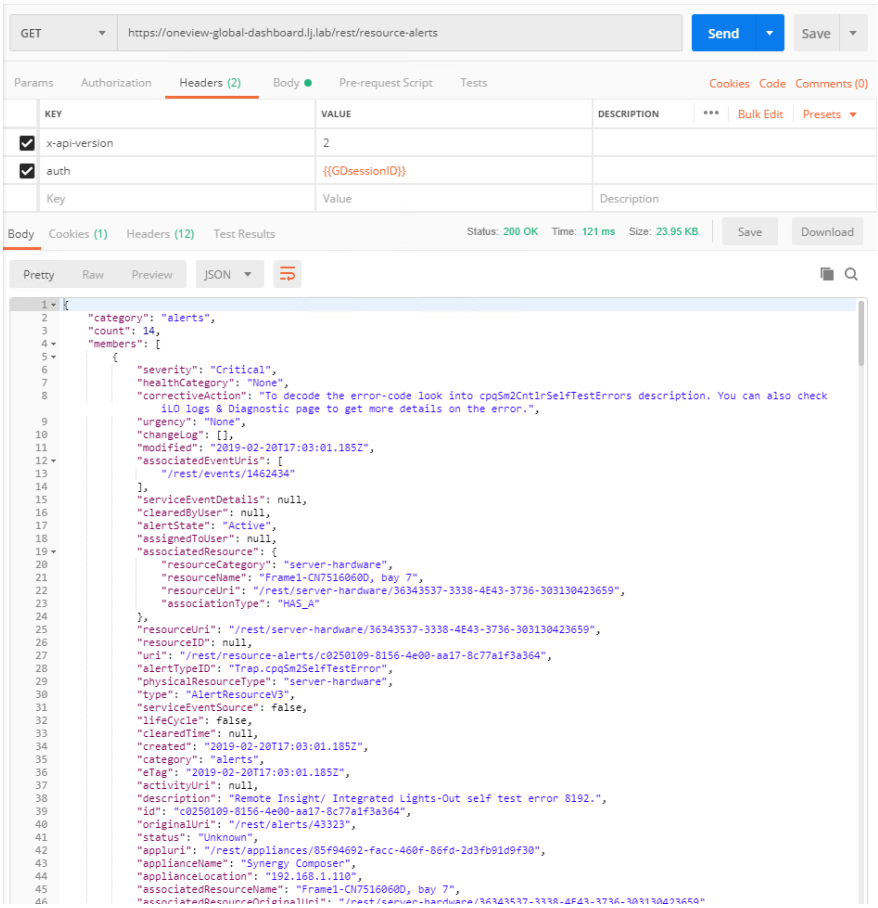
Note: You can remove the *Auth* sessionID and try again as a quick check, you should get a 401 Unauthorized status code.

The most interesting data that can be found in Global Dashboard is the resource alerts. Global Dashboard collects all of the critical alerts from the appliances it monitors. Using the REST API and some simple scripting, we can develop a script that would print out all alerts.

So, let's see now what we can get from the resource alerts, in Postman, enter the following parameters:

Field	**Value**
URL	`https://oneview-global-dashboard.lj.lab/rest/resource-alerts`
Verb	`GET`
Header	`Auth : your sesionID` `X-API-Version:2`

You should get a list of alerts received by Global Dashboard from the two appliances it monitors:



This concludes our Lab 6.

Lab 6 provided a quick overview of the HPE OneView Global Dashboard API, we discovered that the approach is identical as the HPE OneView API, we use the same techniques, the same API patterns and like in OneView, the help in the documentation is really helpful.

In the next lab, we will see how we can use scripting language like PowerShell and Python to interact with these APIs.

Lab 7: Scripting languages

To conclude these labs, we can explore scripting languages and see how we can create scripts to execute REST calls and eventually automate their execution.

Like with Postman, scripting languages can be used to interact with the OneView and Global Dashboard REST APIs but they provide much more benefits, you can automate the execution of tasks, the scripts can be combined into more complex programs and provide lot of flexibility and agility.

Using the REST API and some simple scripting, we are going to build in the lab, a script that would print out the number of alerts available in HPE Global Dashboard with a description of alerts, the severity and the associated corrective actions. This exercise can be helpful to see how someone could easily create extremely useful tools for anyone troubleshooting at a data center.

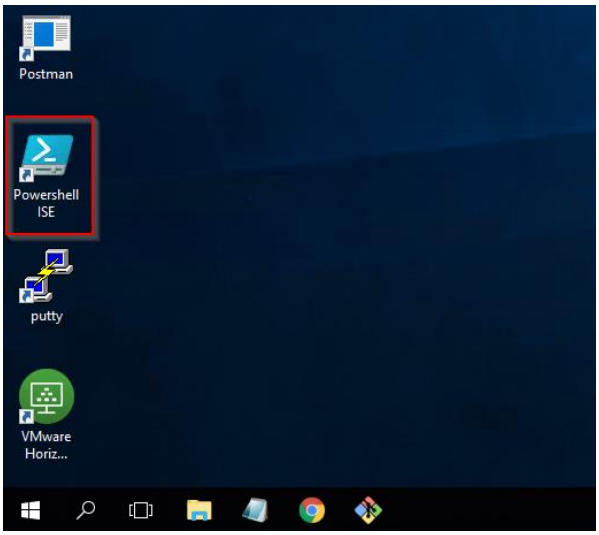
In this lab, we will only glance at two languages, PowerShell and Python. Feel free to jump to the scripting language of your choice.

PowerShell

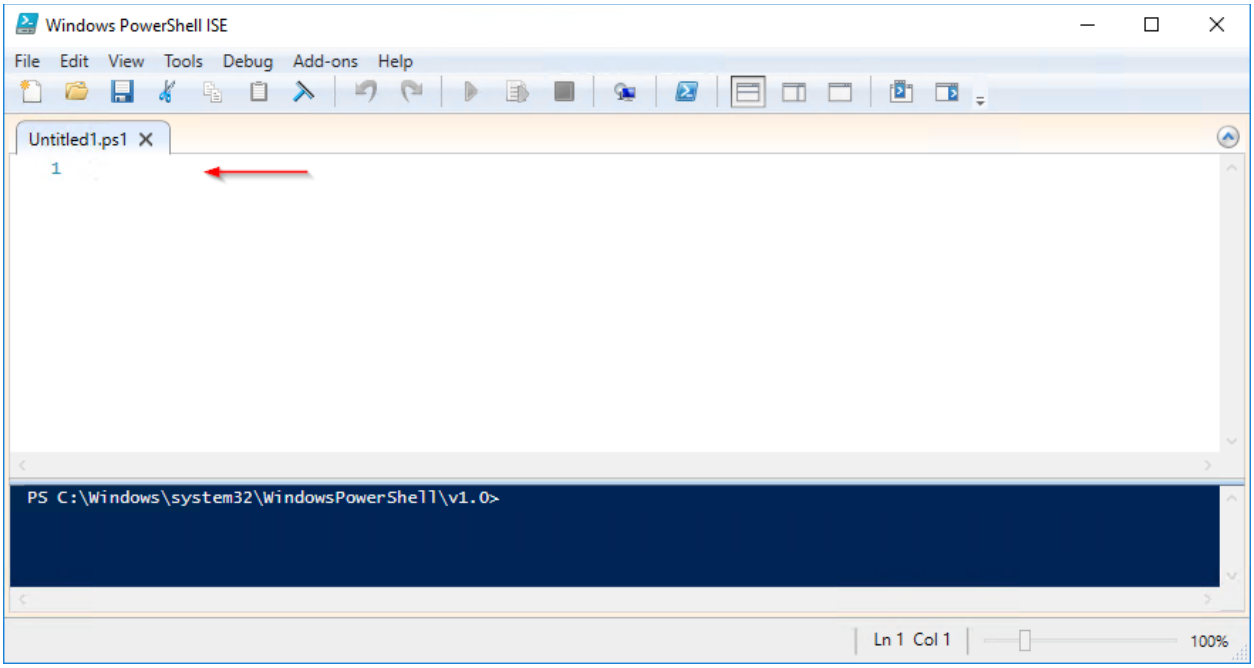
A PowerShell library for HPE OneView is available from HPE, it provides many functions to manage the appliance. We are not going to use this library in this lab as we want you to understand the way a REST call can be realized in PowerShell. Besides that, a library cannot provide all REST calls provided by an API so it is essential to know how to pass a REST request natively in PowerShell.

For more information about the HPE OneView PowerShell library, see https://hewlettpackard.github.io/POSH-HPOneView/

Open PowerShell ISE using the shortcut on the desktop:



Then copy/paste the following script in the script section of PowerShell ISE:

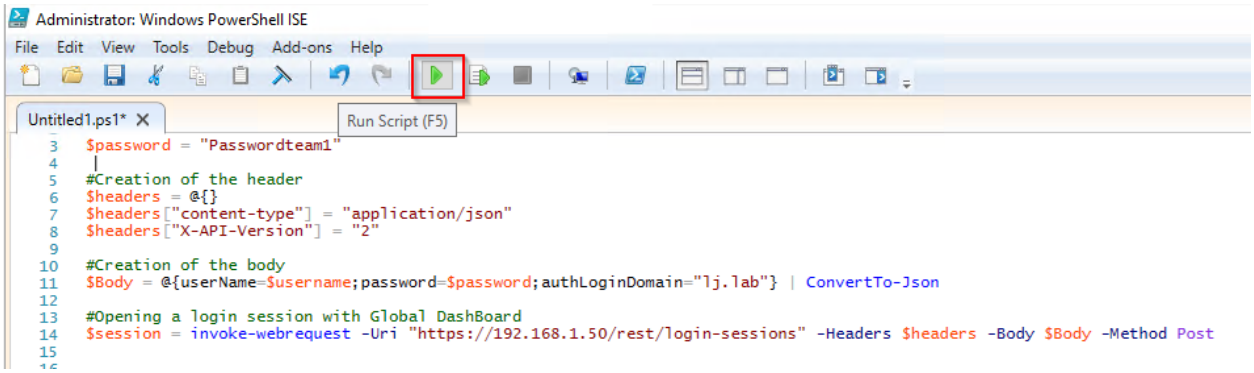


Note: the script pane can be opened by pressing <CRTL> + <R>

```
#Defining the Global Dashboard credentials
$username = "team1"
$password = "Passwordteam1"

#Creation of the header
$headers = @{}
$headers["content-type"] = "application/json"
$headers["X-API-Version"] = "2"

#Creation of the body
$Body = @{userName=$username;password=$password;authLoginDomain="lj.lab"} | ConvertTo-Json

#Opening a login session with Global DashBoard
$session = invoke-webrequest -Uri "https://192.168.1.50/rest/login-sessions" -Headers $headers -
Body $Body -Method Post
```

Notice that the script follows the same steps we saw previously with Postman: the creation of the header (with `content-type` and `x-api-version`), the creation of the body for the login session (with `userName`/`password`/`authLoginDomain`) then we use the PowerShell cmdlet `invoke-webrequest`. This cmdlet is the PowerShell command to send HTTP/HTTPS/FTP and FILE request to a web page or web service. In our case, we will use it for a HTTPS request to our Global Dashboard REST API using a `POST` method on `\rest\login-sessions`. The *URI*, *headers*, *body* and *method* are provided as well as parameters.

Once properly set, execute the script by pressing on the **Run Script** button



The script should run without error like seen below:

```
PS C:\Users\team1.lj>  #Defining the Global Dashboard credentials
$username = "team1"
$password = "Passwordteam1"

#Creation of the header
$headers = @{}
$headers["content-type"] = "application/json"
$headers["X-API-Version"] = "2"

#Creation of the body
$Body = @{userName=$username;password=$password;authLoginDomain="lj.lab"} | ConvertTo-Json

#Opening a login session with Global DashBoard
$session = invoke-webrequest -Uri "https://192.168.1.50/rest/login-sessions" -Headers $headers -Body $Body -Method P


PS C:\Users\team1.lj>
```

To get the result of our request, we can retrieve the value of `$session`, the variable we used to store the result of our `invoke-webrequest` command, to do so, enter in the console `$session` then press **enter**:

```
PS C:\Users\team1.lj> $session


StatusCode        : 200
StatusDescription : OK
Content           : {"user":{"userName":"team1","user_roles":["Infrastructure administrator"],"domain":"lj.lab"},
                    "token":"MzI2MTczODQyODM2ZkG1ASz3ubtzqUsVdtl5uOf8OvjqxuNb","sessionID":"MzI2MTczODQyODM2ZkG1A
                    Sz3ubtzqUsVdtl...
RawContent        : HTTP/1.1 200 OK
                    content-language: en-us
                    Vary: Accept-Encoding
                    Content-Length: 215
                    Cache-Control: no-cache, private
                    Content-Type: application/json; charset=utf-8
                    Date: Fri, 22 Feb 2019 13:35:45 G...
Forms             : {}
Headers           : {[content-language, en-us], [Vary, Accept-Encoding], [Content-Length, 215], [Cache-Control,
                    no-cache, private]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 215
```

As we can see, the result holds several property fields, like a *statuscode*, a *statusdescription* and more importantly a *content* property.

To access the *content* property, it's easy, just enter:
`$session.content`

```
PS C:\Users\team1.lj>
PS C:\Users\team1.lj> $session.Content
{"user":{"userName":"team1","user_roles":["Infrastructure administrator"],"domain":"lj.lab"},"token":"MzI2MTczODQy
ODM2ZkG1ASz3ubtzqUsVdtl5uOf80vjqxuNb","sessionID":"MzI2MTczODQyODM2ZkG1ASz3ubtzqUsVdtl5uOf80vjqxuNb"}
```

Notice that the content of *content* is in JSON format.

With PowerShell, we can easily convert JSON data in something easier to manipulate by using `convertfrom-json` cmdlet, enter:

`$session.Content | ConvertFrom-Json`

```
PS C:\Users\team1.lj>
PS C:\Users\team1.lj> $session.Content | ConvertFrom-Json

user                                                token                                                ses
                                                                                                         sio
                                                                                                         nID
----                                                -----                                                ---
@{userName=team1; user_roles=System.Object[]; domain=lj.lab} MzI2MTczODQyODM2ZkG1ASz3ubtzqUsVdtl5uOf80vjqxuNb MzI
```

The result is now structured as a PowerShell custom object, an object that can be manipulated using properties. Notice the presence of the *token* property, the session token provided by the Global Dashboard API, this is what we are looking for. An easy way to retrieve the *sessionID* is to enter:

`($session.Content | ConvertFrom-Json).token`

```
PS C:\Users\team1.lj>
PS C:\Users\team1.lj> ($session.Content | ConvertFrom-Json).token
MzI2MTczODQyODM2ZkG1ASz3ubtzqUsVdtl5uOf80vjqxuNb

PS C:\Users\team1.lj>
```

Now you can enter the following lines and press enter:

```
#Capturing the OneView Global DashBoard Session ID and adding it to the header
$key = ($session.content | ConvertFrom-Json).token
$headers["auth"] = $key
```

We use the variable `$key` to store the value of the *sessionID* token. Then we add that `$key` to the header.

We can now use this *auth* token for all our API call methods.

```
PS C:\Users\team1.lj> $key = ($session.content | ConvertFrom-Json).token
$headers["auth"] = $key
```

As an example, we can use the `GET /rest/resource-alerts` method to retrieve the critical alerts received by Global Dashboard, we will use once again the invoke-webrequest, enter now:

```
#Capturing the resource alerts
$resourcealerts = (invoke-webrequest -Uri "https://192.168.1.50/rest/resource-alerts" -Headers
$headers -Method GET) | ConvertFrom-Json
```

Note: make sure you type it as a single line

```
PS C:\Users\team1.lj> $resourcealerts = (invoke-webrequest -Uri "https://192.168.1.50/rest/resource-alerts" -Headers
PS C:\Users\team1.lj> |
```

The result is saved in `$resourcealerts`

You can retrieve the content of $resourcealerts by entering:

```
$resourcealerts | Get-Member
```

```
PS C:\Users\team1.lj> $resourcealerts | Get-Member


   TypeName: System.Management.Automation.PSCustomObject

Name         MemberType   Definition
----         ----------   ----------
Equals       Method       bool Equals(System.Object obj)
GetHashCode  Method       int GetHashCode()
GetType      Method       type GetType()
ToString     Method       string ToString()
category     NoteProperty string category=alerts
count        NoteProperty int count=12
members      NoteProperty Object[] members=System.Object[]
nextPageUri  NoteProperty object nextPageUri=null
prevPageUri  NoteProperty object prevPageUri=null
start        NoteProperty int start=0
total        NoteProperty int total=12
uri          NoteProperty string uri=/rest/resource-alerts?count=25&start=0&query='severity%20EQ%20%22Cri
```

`Get-member` is a PowerShell function which returns the members, the properties and methods of objects. And as we can see, the result holds several property fields, like *uri*, *category*, etc. and more importantly a *members* property.

To access the *members* property, it's easy, just enter:
`$resourcealerts.members`

The list can be very long and not easy to read so let's work on way to get a better presentation. We can retrieve the members of this object using this time:

```
$resourcealerts.members | Get-Member
```

```
PS C:\Users\team1.lj> $resourcealerts.members | Get-Member


   TypeName: System.Management.Automation.PSCustomObject

Name                          MemberType   Definition
----                          ----------   ----------
Equals                        Method       bool Equals(System.Object obj)
GetHashCode                   Method       int GetHashCode()
GetType                       Method       type GetType()
ToString                      Method       string ToString()
activityUri                   NoteProperty string activityUri=/rest/tasks/38804583-9433-4832-B3AD-FCF9C0FF17AF
alertState                    NoteProperty string alertState=Locked
alertTypeID                   NoteProperty string alertTypeID=server-hardware.mp.config.ipmanagerblob.add
applianceLocation             NoteProperty string applianceLocation=192.168.1.10
applianceModel                NoteProperty string applianceModel=HPE OneView VM - VMware vSphere
applianceName                 NoteProperty string applianceName=HPE OneView
applianceVersion              NoteProperty string applianceVersion=4.10.04-0370820
appluri                       NoteProperty string appluri=/rest/appliances/ca3758a0-d368-4e26-8b18-6c53b50eef5f
assignedToUser                NoteProperty object assignedToUser=null
associatedEventUris           NoteProperty Object[] associatedEventUris=System.Object[]
associatedResource            NoteProperty System.Management.Automation.PSCustomObject associatedResource=@{resourceUri=/rest/
associatedResourceName        NoteProperty string associatedResourceName=iLO-ESX5-0
associatedResourceOriginalUri NoteProperty string associatedResourceOriginalUri=/rest/server-hardware/31373736-3939-4D32-3431-
category                      NoteProperty string category=alerts
changeLog                     NoteProperty Object[] changeLog=System.Object[]
clearedByUser                 NoteProperty object clearedByUser=null
clearedTime                   NoteProperty object clearedTime=null
correctiveAction              NoteProperty string correctiveAction=Ensure network connectivity to the iLO is available, reset
created                       NoteProperty string created=2019-02-23T15:39:32.269Z
description                   NoteProperty string description=Unable to register this HPE OneView instance with iLO: There was
eTag                          NoteProperty string eTag=2019-02-23T15:39:32.269Z
healthCategory                NoteProperty string healthCategory=Operational
id                            NoteProperty string id=fc41fb42-7766-49eb-8c9f-5b89eb1d6da6
lifeCycle                     NoteProperty bool lifeCycle=False
modified                      NoteProperty string modified=2019-02-23T15:39:32.269Z
originalUri                   NoteProperty string originalUri=/rest/alerts/1129470
physicalResourceType          NoteProperty string physicalResourceType=server-hardware
resourceID                    NoteProperty object resourceID=null
resourceUri                   NoteProperty string resourceUri=/rest/server-hardware/31373736-3939-4D32-3431-333034374B4B
serviceEventDetails           NoteProperty object serviceEventDetails=null
serviceEventSource            NoteProperty bool serviceEventSource=False
severity                      NoteProperty string severity=Critical
status                        NoteProperty string status=Unknown
type                          NoteProperty string type=AlertResourceV3
urgency                       NoteProperty string urgency=None
uri                           NoteProperty string uri=/rest/resource-alerts/fc41fb42-7766-49eb-8c9f-5b89eb1d6da6
```

These properties highlighted above are all the properties available for this object, some are very useful for what we are looking for, some others are not.

So, let's work on the formatting of this object to display only the important information we need. We are going to display only the severity, the description and the corrective action for these alerts by selecting only the appropriate properties, enter:

```
$resourcealerts.members | Format-List -Property severity,description,correctiveAction
```

The display result is much better now and easier to read:

```
severity         : Critical
description      : The hypervisor manager
                   {"name":"vcenter.lj.mougins.net","uri":"/rest/hypervisor-managers/1dacb777-ff84-41c2-937f-c733cc1445fd"} is disconnected.
correctiveAction : Refer to the hypervisor manager
                   {"name":"vcenter.lj.mougins.net","uri":"/rest/hypervisor-managers/1dacb777-ff84-41c2-937f-c733cc1445fd"} for more details.

severity         : Critical
description      : The hypervisor manager is not reachable.
correctiveAction : Check network connectivity to the hypervisor manager.  Alternatively, if hypervisor manager settings are changed then edit the
                   hypervisor manager resource in HPE OneView and specify valid IPv4 address or host name, and port.

severity         : Critical
description      : The state of the server hardware in iLO-ESX5-0 is Unmanaged, but needs to be Profile Applied in order to be operational.
correctiveAction : Refer to the server hardware {"name":"iLO-ESX5-0", "uri":"/rest/server-hardware/31373736-3939-4D32-3431-333034374B4B"} for
                   more information on how to fix the issue or edit the server profile and select different server hardware.

severity         : Critical
description      : All of the managed storage target ports for the storage system are non-operational.
correctiveAction : Examine the alerts related to the individual ports and resolve them.

severity         : Critical
description      : The firmware baseline {"name":"HP Service Pack for ProLiant version 2015.10.0",
                   "uri":"/rest/firmware-drivers/843216_001_spp_2015_10_0-SPP2015100_2015_0921_6"} has been removed.
correctiveAction : Edit the server profile template and select a different firmware baseline, or choose to manage firmware manually.
```

Lastly, we can display the number of Global Dashboard alerts by entering:

```
Write-host "The number of alerts is" $resourcealerts.count
```

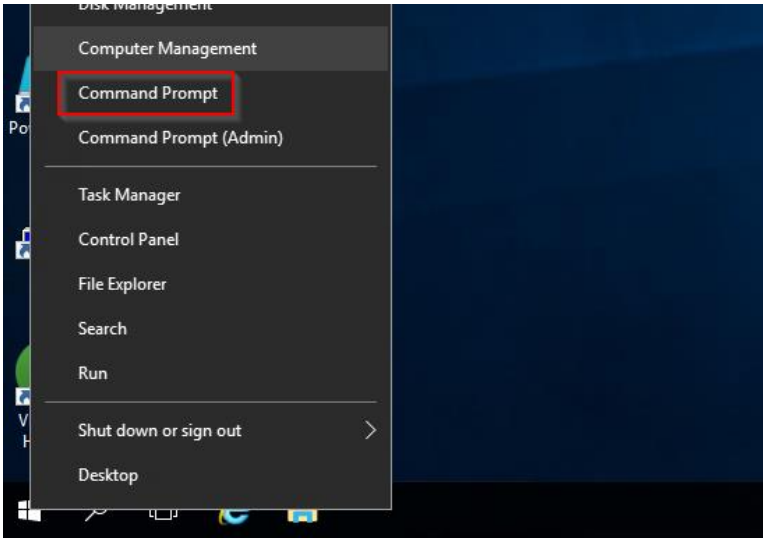
```
PS C:\Users\team1.lj> Write-host "`nThe number of alerts is" $resourcealerts.count

The number of alerts is 12
```

Python

A Python library for HPE OneView is available from HPE, it provides basic functions to manage the appliance. We are not going to use this library in this lab as we want you to understand the way a REST call can be realized in Python. Besides that, a library cannot provide all REST calls provided by an API so it is essential to know how to pass a REST request natively in python.

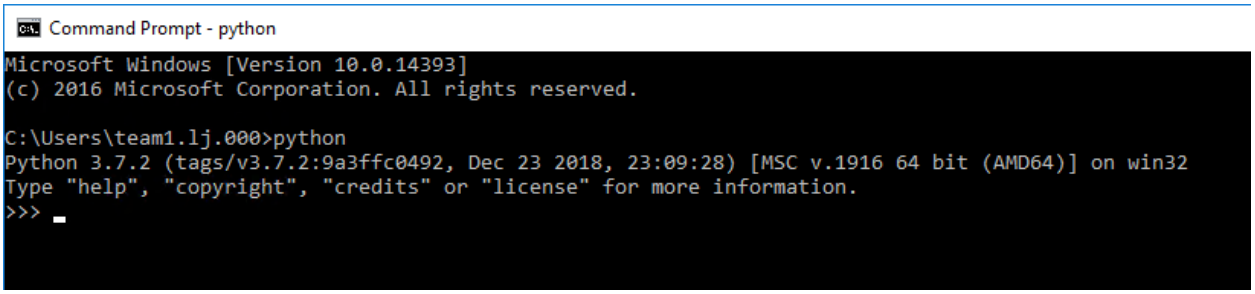
For more information about the HPE OneView Python library, see https://github.com/HewlettPackard/python-hpOneView

Open Command Prompt using right click on windows start on the desktop:



Then launch python by entering:

python



Enter one by one the following lines and press **Enter** after each line to check the execution or copy/paste the all script to save time:

```
# Importing the required libraries
import requests

# Defining the Global Dashboard credentials
username = "team1"
password = "Passwordteam1"

# Creating the body content
body = {
          'userName': username,
             'password': password,
             'authLoginDomain': 'lj.lab'
        }

# Creating the header content
headers={
                      'Content-Type': 'application/json',
                      'X-API-VERSION': '2'
                     }

# Creating the web request with body and header content

response = requests.post(

        url= "https://192.168.1.50/rest/login-sessions",
        verify=False,
        json=body,
        headers=headers
        )
```

Notice that the script follows the same steps we saw previously with Postman: the creation of the body for the login session (with *userName/password/authLoginDomain*), the creation of the header (with *content-type* and *x-api-version*), then the API query using the Python `requests` function. This function is imported at the beginning of the script from the Python library *Requests* used to send HTTP/HTTPS/FTP and FILE request to a web page or web service. We will use it here for a HTTPS request to our Global Dashboard REST API using a `POST` method on `\rest\login-sessions`. The *URL*, *headers*, *body* are provided as well as parameters.

The script should run without error like seen below:

```
...             'password': password,
...             'authLoginDomain': 'lj.lab'
...         }
>>> #print(body)
... # Creating the header content
...
>>> headers={
...                         'Content-Type': 'application/json',
...                         'X-API-VERSION': '2'
...                        }
>>>  #print(header)
... # Creating the web request with body and header content
...
>>> response = requests.post(
...         url= "https://192.168.1.50/rest/login-sessions",
...         verify=False,
...         json=body,
...         headers=headers
...         )
C:\Program Files\Python37\lib\site-packages\urllib3\connectionpool.py:847: InsecureRequestWarning: Unverified HTTPS
ised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning)
>>> # Converting json response into python dictionary
...
>>> loginsessionresponse = json.loads(response.content)
>>> #print(json.dumps(loginsessionresponse, indent=4))
... # Collecting sessionID from the response
...
>>> sessionid = loginsessionresponse['sessionID']
>>>
>>>
```

You can retrieve the content of `response` by entering:

```
>>> dir(response)
```

```
>>> dir(response)
['__attrs__', '__bool__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__enter__', '__eq__', '__ex
it__', '__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclas
s__', '__iter__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__nonzero__', '__reduce__', '__reduce_ex__
', '__repr__', '__setattr__', '__setstate__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_content'
, '_content_consumed', '_next', 'apparent_encoding', 'close', 'connection', 'content', 'cookies', 'elapsed', 'encodi
ng', 'headers', 'history', 'is_permanent_redirect', 'is_redirect', 'iter_content', 'iter_lines', 'json', 'links', 'n
ext', 'ok', 'raise_for_status', 'raw', 'reason', 'request', 'status_code', 'text', 'url']
>>>
```

`dir()` is an inbuilt Python function which returns list of the attributes and methods of any object. And as we can see, the result holds several property fields, like a *reason*, a *status_code* and more importantly a *content* property.

To access the *content* property, it's easy, just enter:
`>>> response.content`

`response` is the variable we are using to store the result of `requests.post`

```
>>> response.content
b'{"user":{"userName":"team1","user_roles":["Infrastructure administrator"],"domain":"lj.lab"},"token":"LTYwOTA2OTcx
MzgyOhOcAKB_SnAuE9fX1dXnIy4wPubuftCy","sessionID":"LTYwOTA2OTcxMzgyOhOcAKB_SnAuE9fX1dXnIy4wPubuftCy"}'>>>
```

Notice that the content of *content* is in JSON format.

With Python, we can easily convert JSON data in something easier to manipulate by using json library, enter the following to import the library:

```
>>> import json
```

Then to convert the json response into Python dictionary, enter:

```
>>> loginsessionresponse = json.loads(response.content)
```

The result is now structured as a Python dictionary. A Python dictionary is a mapping of unique keys to values that can be easily manipulated and accessed.

To get a nice looking of `loginsessionresponse`, enter:

```
>>> print(json.dumps(loginsessionresponse, indent=4))
```

```
>>> print(json.dumps(loginsessionresponse, indent=4))
{
    "user": {
        "userName": "team1",
        "user_roles": [
            "Infrastructure administrator"
        ],
        "domain": "lj.lab"
    },
    "token": "LTYwOTA2OTcxMzgyOhOcAKB_SnAuE9fX1dXnIy4wPubuftCy",
    "sessionID": "LTYwOTA2OTcxMzgyOhOcAKB_SnAuE9fX1dXnIy4wPubuftCy"
}
>>>
```

Notice the presence of the *sessionID* property, the session token provided by the Global Dashboard API, this is what we are looking for.

An easy way to retrieve the *sessionID* from the dictionary is to enter:

```
>>> loginsessionresponse['sessionID']
```

```
>>> loginsessionresponse['sessionID']
'LTYwOTA2OTcxMzgyOhOcAKB_SnAuE9fX1dXnIy4wPubuftCy'
>>>
```

Then we can enter:

```
>>> sessionid = loginsessionresponse['sessionID']
```

to save the Global Dashboard session ID into the variable `sessionID`.

Now every Global Dashboard API call method we do will use this sessionID token for the authentication, so if we want for instance get the resource alerts, we can enter the following lines and press enter:

```
response_alerts = requests.get(
    url="https://192.168.1.50/rest/resource-alerts",
    verify=False,
    headers={
           'Content-Type': 'application/json',
           'X-API-VERSION': '2',
           'auth': sessionid
          }
    )
```

We use the variable `sessionid` to store the value of the authentication token. Then we use it in the header with `auth`. We also use the `verify=False` option to avoid a certificate verification failure as we are using a self-signed certificate in Global Dashboard.

In this example, we use the `GET /rest/resource-alerts` method to retrieve the critical alerts received by Global Dashboard.

To get the list of alerts, we can enter:

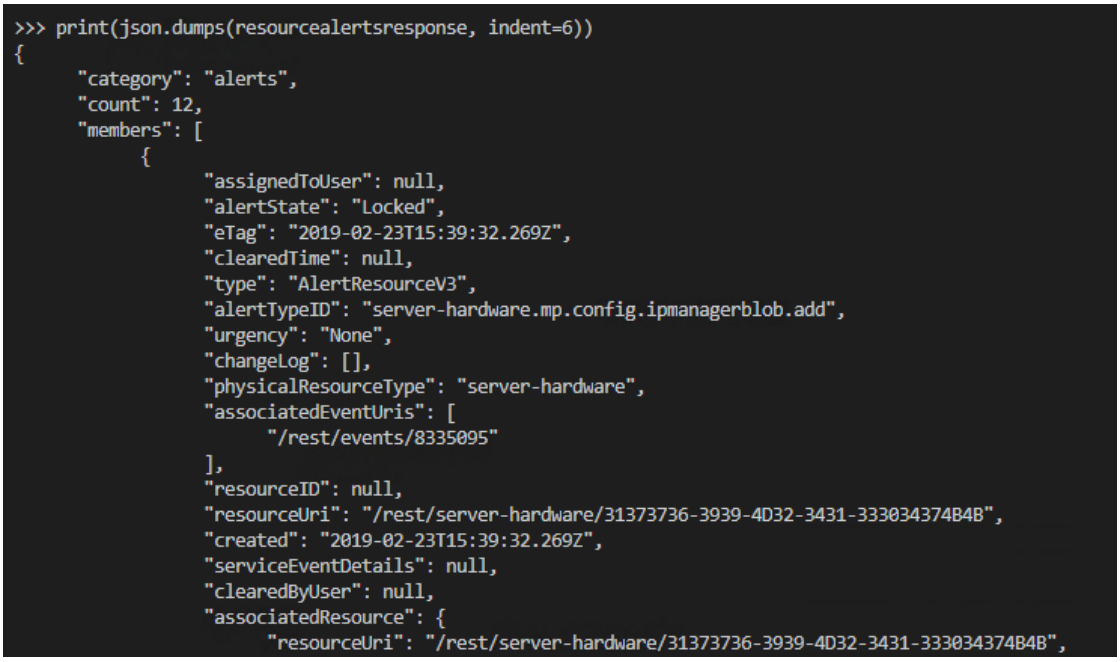
```
>>> resourcealertsresponse = json.loads(response_alerts.content)
```

The result is saved in `resourcealertsresponse`

To display the content of `resourcealertsresponse`, enter:

```
>>> print(json.dumps(resourcealertsresponse, indent=6))
```

`json.dumps` is used here to format the content of the json response into something more readable.



```
>>> print(json.dumps(resourcealertsresponse, indent=6))
{
      "category": "alerts",
      "count": 12,
      "members": [
            {
                  "assignedToUser": null,
                  "alertState": "Locked",
                  "eTag": "2019-02-23T15:39:32.269Z",
                  "clearedTime": null,
                  "type": "AlertResourceV3",
                  "alertTypeID": "server-hardware.mp.config.ipmanagerblob.add",
                  "urgency": "None",
                  "changeLog": [],
                  "physicalResourceType": "server-hardware",
                  "associatedEventUris": [
                        "/rest/events/8335095"
                  ],
                  "resourceID": null,
                  "resourceUri": "/rest/server-hardware/31373736-3939-4D32-3431-333034374B4B",
                  "created": "2019-02-23T15:39:32.269Z",
                  "serviceEventDetails": null,
                  "clearedByUser": null,
                  "associatedResource": {
                        "resourceUri": "/rest/server-hardware/31373736-3939-4D32-3431-333034374B4B",
```

From the result, we see that the alerts are grouped into a list, sometimes referred as a collection, identified by the [...] (1). This list is called *members*. The { symbol (2) means the list is composed of dictionaries (paired list of items, `"items": "value"`).


```
>>> print(json.dumps(resourcealertsresponse, indent=6))
{
      "category": "alerts",
      "count": 12,
      "members": [
            {
                  "assignedToUser": null,
                  "alertState": "Locked",
                  "eTag": "2019-02-23T15:39:32.269Z",
                  "clearedTime": null,
                  "type": "AlertResourceV3",
                  "alertTypeID": "server-hardware.mp.config.ipmanagerblob.add",
                  "urgency": "None",
                  "changeLog": [],
                  "physicalResourceType": "server-hardware",
```

So, we can reduce the result to only the alerts by entering:

```
>>> alerts = resourcealertsresponse['members']
```

Then we can format the alerts by using:

```
>>> print(json.dumps(alerts, indent=6))
```


```
>>> print(json.dumps(alerts, indent=6))
[
      {
            "assignedToUser": null,
            "alertState": "Locked",
            "eTag": "2019-02-23T15:39:32.269Z",
            "clearedTime": null,
            "type": "AlertResourceV3",
            "alertTypeID": "server-hardware.mp.config.ipmanagerblob.add",
            "urgency": "None",
            "changeLog": [],
            "physicalResourceType": "server-hardware",
            "associatedEventUris": [
                  "/rest/events/8335095"
            ],
            "resourceID": null,
            "resourceUri": "/rest/server-hardware/31373736-3939-4D32-3431-333034374B4B",
            "created": "2019-02-23T15:39:32.269Z",
            "serviceEventDetails": null,
            "clearedByUser": null,
            "associatedResource": {
                  "resourceUri": "/rest/server-hardware/31373736-3939-4D32-3431-333034374B4B",
                  "associationType": "HAS_A",
                  "resourceCategory": "server-hardware",
                  "resourceName": "iLO-ESX5-0"
            },
            "serviceEventSource": false,
            "category": "alerts",
            "correctiveAction": "Ensure network connectivity to the iLO is available, reset the iLO using iLO web interface, wait for
 iLO to restart and refresh the server. If the problem persists, power cycle the server, wait for the server to complete Power On Sel
f-Test (POST) and then refresh the server. If there are any active alerts for the
server that are related to certificate based communication with the iLO, resolve the issue as per the resolution steps  provided in t
he alert and retry the operation.",
            "healthCategory": "Operational",
            "uri": "/rest/resource-alerts/fc41fb42-7766-49eb-8c9f-5b89eb1d6da6",
```


Because members is a list, we can get the first alert by entering [0]:

```
>>> print(json.dumps(alerts[0], indent=6))
```

And we can display the number of alerts using:

```
>>> print("The number of alerts is " + str(len(alerts)))
```

```
>>> print("The number of alerts is " + str(len(alerts)))
The number of alerts is 12
>>>
```

And finally, to display all alerts in a nice format, enter:

```
for alert in alerts:
  print("\nSeverity: " + alert['severity'])
  print("Description: " + alert['description'])
  print("Corrective Action: " + alert['correctiveAction'])
```

WARNING: do not copy/paste the 4 lines above from the PDF version of this guide as this will lose the tab indentation that Python relies on for its syntax. You can copy/paste the first line, hit enter in the console, then copy/paste the other lines.

This loop is used to display all alerts in the list, it displays the following result:

```
Severity: Critical
Description: Unable to register this HPE OneView instance with iLO: There was a problem with posting a command to the iLO..
Corrective Action: Ensure network connectivity to the iLO is available, reset the iLO using iLO web interface, wait for iLO
he server to complete Power On Self-Test (POST) and then refresh the server. If there are any active alerts for the server
resolution steps  provided in the alert and retry the operation.

Severity: Critical
Description: Unable to register this HPE OneView instance with the iLO because the NAND flash device was not initialized or
Corrective Action: A full description and corrective actions can be found in the following advisory:

http://h20564.www2.hpe.com/hpsc/doc/public/display?docId=emr_na-c04996097

Severity: Critical
Description: Communication with the SAN manager has failed or timed out.
Corrective Action: Check the network connectivity between the appliance and the SAN manager, the correct port,SNMP v3 crede
 if the device is supported per the support matrix.

Severity: Critical
Description: A SAN switch is not accessible (Brocade_4).
Corrective Action: Check that the SAN's switches are visible to the SAN Manager and that their status is healthy.
```

Note: `\n` is used to insert a space (a newline) between each alert for better clarity

Summary

During this lab, you have been introduced to the HPE OneView and HPE OneView Global Dashboard RESTful API. You have learned about REST API concepts, how to login and start to leverage some of the API methods, how to find and decrypt the right resource method and how the OneView resource model is built. You also learned how to use Postman and how to decrypt JSON responses. Finally, you discovered how to use PowerShell and Python to send web request tasks against the REST API, how to translate the JSON response and how to manipulate objects.

If you want to go further and automate your infrastructure management and eliminate complex manual processes, I would recommend you to visit the HPE OneView Composable Infrastructure Ecosystem, see https://www.hpe.com/us/en/solutions/developers/composable.html