

# **HPE Composable Infrastructure Hands-On Lab**

Programing HPE OneView and HPE Global  
Dashboard



## Contents

Programming HPE OneView and HPE Global Dashboard Hands-On Lab .....	5
Summary.....	5
REST API.....	5
Lab 1: Read the Fantastic Manual .....	7
Lab 2: REST Tool.....	19
Lab 3: Login Sessions.....	24
Lab 4: API Version and backward compatibility.....	31
Lab 5: API Resource model .....	35
Lab 6: HPE Global Dashboard API .....	55
Lab 7: Scripting languages.....	68
PowerShell.....	68
Python.....	75
Summary.....	83



### Programming 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](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
<b>Create</b>	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.
<b>Read</b>	GET URI	Returns the requested resource representation(s)
<b>Update</b>	PUT URI <Payload = Update data>	Update an existing resource using the update data.
<b>Delete</b>	DELETE URI	Delete the addressed resource

### **URI format**

All the appliance URLs point to resources and the client does not need to modify or create URLs. The URL 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

<code>https://{appl}</code>	.	The appliance address
<code>/rest</code>		Type of URI.
<code>/{resource name}</code>		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](http://www.json.org)



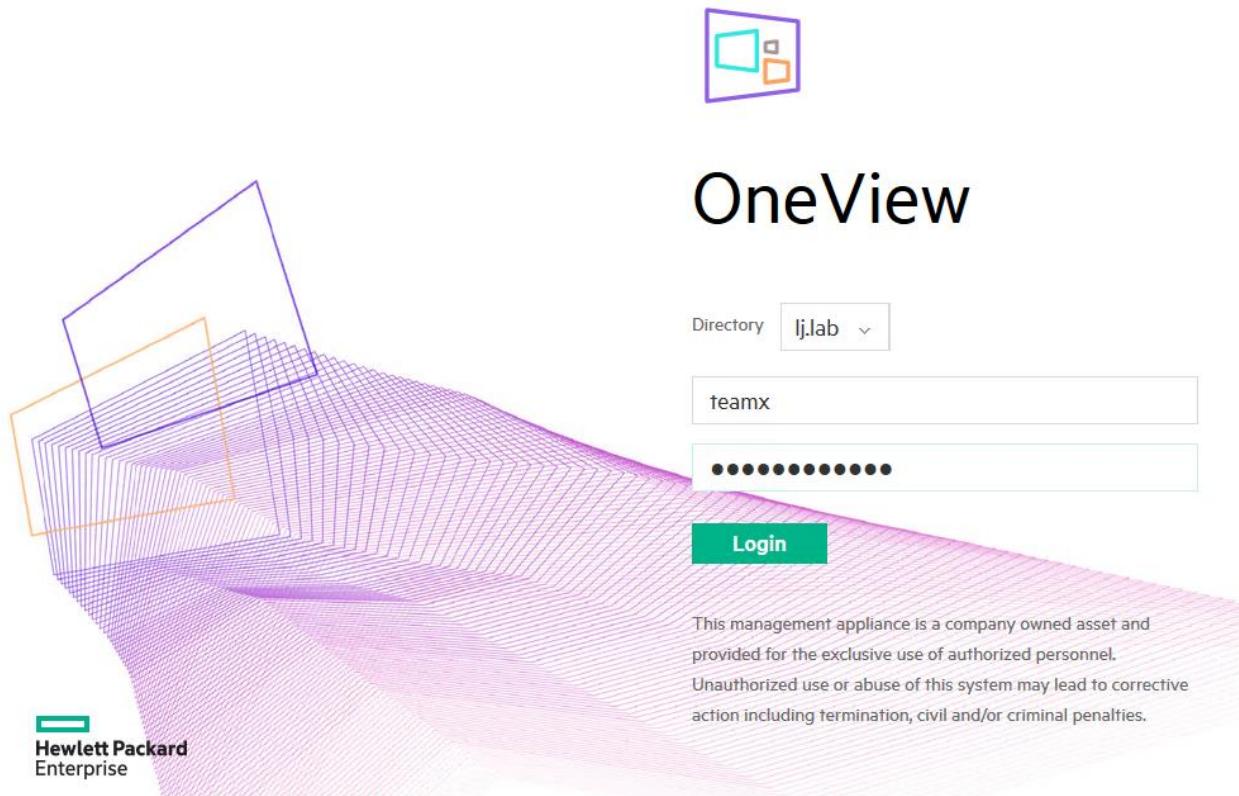
### Lab 1: Read the Fantastic Manual

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:

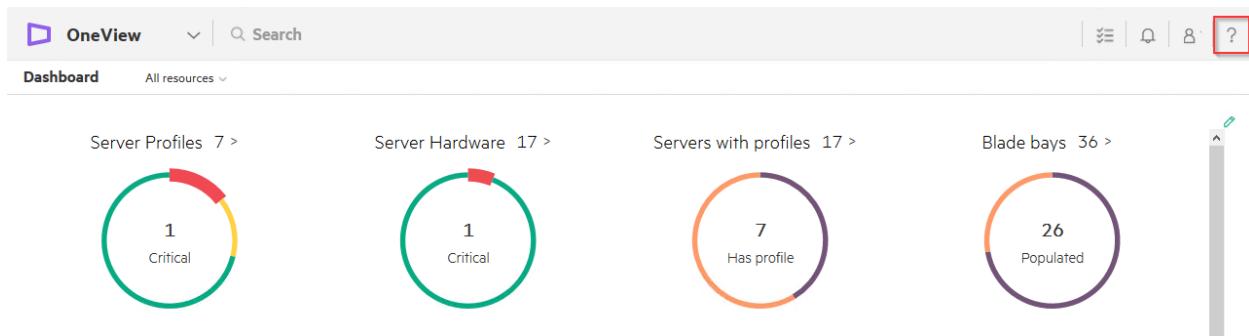
- Directory: **lj.lab**
- Username: **teamx**
- Password : **Passwordteamx**

with **x** being your team number





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



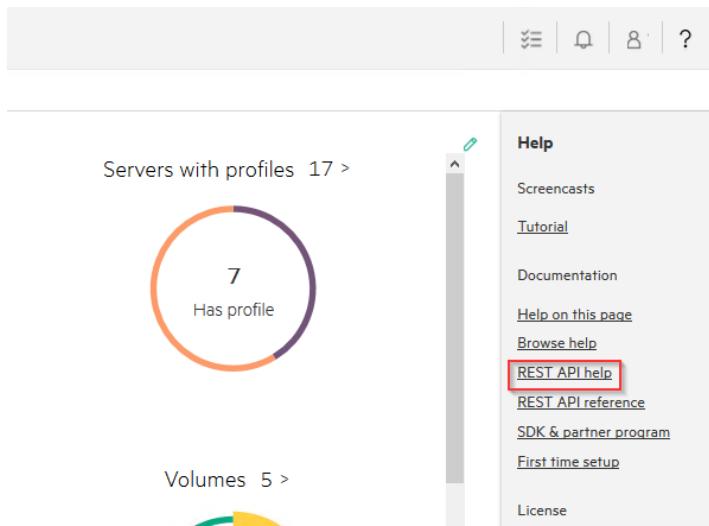
The dashboard displays four main categories:

- Server Profiles: 7 > (1 Critical)
- Server Hardware: 17 > (1 Critical)
- Servers with profiles: 17 > (7 Has profile)
- Blade bays: 36 > (26 Populated)

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 Help menu includes:

- Screencasts
- Tutorial
- Documentation
- Help on this page
- Browse help
- REST API help** (highlighted with a red box)
- REST API reference
- SDK & partner program
- First time setup
- License

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

### 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
<a href="#">Previous Versions of HPE OneView Documents on HPE Support Center</a>	 HTML		
<a href="#">HPE OneView 4.1 Help</a>	 HTML	N/A	Jun 2018
<a href="#">HPE OneView 4.1 Help for HPE Synergy</a>	 HTML	N/A	Jun 2018
<a href="#">HPE OneView 4.1 API Reference (API version 800)</a>	 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?	<a href="#">Learning about REST APIs</a>
What tasks can I perform using the REST APIs?	<a href="#">What can I do with HPE OneView REST APIs?</a>
Where can I find detailed reference information about the REST API resource model schema and JSON examples?	<a href="#">HPE OneView REST API Reference</a>



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?	<a href="#">Learning about REST APIs</a>
What tasks can I perform using the REST APIs?	<a href="#">What can I do with HPE OneView REST APIs?</a>
Where can I find detailed reference information about the REST API resource model schema and JSON examples?	<a href="#">HPE OneView REST API Reference</a>

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

The screenshot shows the HPE OneView dashboard. On the left, there are two main sections: 'Servers with profiles' (17 >) and 'Volumes' (5 >). A circular icon with the number '7' and the text 'Has profile' is highlighted with an orange circle. On the right, a vertical sidebar has a 'Help' menu open. The menu includes options like 'Screencasts', 'Tutorial', 'Documentation', 'Help on this page', 'Browse help', 'REST API help' (which is highlighted with a red box), 'REST API reference' (also highlighted with a red box), 'SDK & partner program', and 'First time setup'.

Click on **HPE OneView REST API Reference**



# Hewlett Packard Enterprise

TSS Paris 2019

HPE OneView API Reference						
	SERVERS	NETWORKING	STORAGE	FC-SANS	FACILITIES	EXTERNAL MANAGERS
About	Connections	Connection Templates	Drive Enclosures	Endpoints	Datacenters	Hypervisor Managers
Security Model	Enclosure Groups	Ethernet Networks	SAS Logical JBOD	Managed SANs	Power Devices	
Response Codes	Enclosures	Fabric Managers	Attachments	Providers	Racks	
Association Names	ID Pools	Fabrics	SAS Logical JBODs	SAN Managers	Unmanaged Devices	
Common Parameters	ID Pools IPv4 Ranges	FC Networks	Storage Pools			
Common Attributes	ID Pools IPv4 Subnets	FCoE Networks	Storage Systems			
What's New?	ID Pools vMAC Ranges	Interconnect Link Topologies	Storage Volume Attachments			
	ID Pools vSN Ranges	Interconnect Types	Volumes			
	ID Pools vWWN Ranges	Interconnects				
	Logical Enclosures	Internal Link Sets				
	Migratable VC	Logical Downlinks				

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.

HPE OneView API Reference									
	SERVERS	NETWORKING	STORAGE	FC-SANS	FACILITIES	EXTERNAL MANAGERS	HYPERVERSORS	DATA SERVICES	ACTIVITY
About	Connections	Connection Templates	Drive Enclosures	Endpoints	Datacenters	Hypervisor Managers	Hypervisor Cluster Profiles	Metric Streaming	Alerts
Security Model	Enclosure Groups	Ethernet Networks	SAS Logical JBOD	Managed SANs	Power Devices			Remote Syslog	Audit Logs
Response Codes	Enclosures	Fabric Managers	Attachments	Providers	Racks		Hypervisor Host Profiles		Events
Association Names	ID Pools	Fabrics	SAS Logical JBODs	SAN Managers	Unmanaged Devices				Reports
Common Parameters	ID Pools IPv4 Ranges	FC Networks	Storage Pools						Tasks
Common Attributes	ID Pools IPv4 Subnets	FCoE Networks	Storage Systems						
What's New?	ID Pools vMAC	Interconnect Link Topologies	Storage Volume Attachments						



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.

**HPE OneView** ▾

**API Reference**

About	SERVERS	NETWORKING
Security Model	Connections	Connection
Response Codes	Enclosure Groups	Templates
Association Names	Enclosures	Ethernet Networks
Common Parameters	ID Pools	Fabric Managers
Common Attributes	ID Pools IPv4 Ranges	Fabrics
What's New?	ID Pools IPv4 Subnets	FC Networks
	ID Pools vMAC	FCoE Networks
	Ranges	Interconnect Link

Navigate rapidly through the different menus to explore their content.

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

**HPE OneView** ▾

**API Reference**

About	SERVERS	NETWORKING	STORAGE	FC-SANS
Security Model	Connections	Connection	Drive Enclosures	Endpoints
Response Codes	Enclosure Groups	Templates	SAS Logical JBOD	Managed SANs
Association Names	<b>Enclosures</b>	Ethernet Networks	Attachments	Providers
Common Parameters	ID Pools	Fabric Managers	SAS Logical JBODs	SAN Managers
Common Attributes	ID Pools IPv4 Ranges	Fabrics	Storage Pools	
What's New?	ID Pools IPv4 Subnets	FC Networks	Storage Systems	
	ID Pools vMAC	FCoE Networks	Storage Templates	



The following is displayed:

The screenshot shows the HPE OneView API Reference interface. The left sidebar has a tree view with nodes like 'About', 'Security Model', 'Response Codes', etc., and a selected node 'Enclosures' which is highlighted with a green background. The main content area has a search bar at the top. Below it, the title 'Enclosures' is shown, followed by a brief description: '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.' Below the description is a list of API methods:

- ▶ **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.
- ▶ **POST**    [`/rest/enclosures`](#)  
Adds a remote enclosure and all the enclosures linked to that enclosure by their frame link modules. The remote enclosures' frame link modules must not be claimed by another appliance. The IP used must be the frame link module's Link Local IPv6 address.  
...
- ▶ **GET**    [`/rest/enclosures/{id}`](#)  
Returns the enclosure with the specified ID, if it exists.
- ▶ **PATCH**    [`/rest/enclosures/{id}`](#)  
Use the PATCH REST API to update specific attributes of a given enclosure resource.  
...
- ▶ **DELETE**    [`/rest/enclosures/{id}`](#)  
Removes the specified enclosure from the appliance. All components of the enclosure (e.g. servers and interconnects) are removed. An enclosure can only be removed if it is in the monitored state and connectivity has been lost to the enclosure. To remove a managed enclosure, the logical  
...
- ▶ **PUT**    [`/rest/enclosures/{id}/configuration`](#)  
Reapplies the appliance's configuration on the enclosure. This includes running the same configure steps that were performed as part of the enclosure add. A [Task](#) URI is returned in the Location response header. Use it to track the status and progress of the operation.

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:

```
GET https://{{appl}}/rest/enclosures?filter="status='OK'"  
  
Auth: abcdefghijklmnopqrstuvwxyz012345  
X-Api-Version: 800
```

← 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

<b>Accept-Language</b>	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
<b>Auth</b>	Session authorization token obtained from <a href="#">logging in</a> . If this header is not included or if the session-token is invalid, the response code will be 401 Unauthorized.	string required
<b>X-Api-Version</b>	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

<b>count</b>	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.	integer
See the <a href="#">Common Parameters</a> page for more information on the usage of count.		
<b>Default</b>	-1	
<b>fields</b>	Specifies which fields should be returned in the result set.	string
See the <a href="#">Common Parameters</a> page for more information on the usage of filter.		
<b>filter</b>	A general filter/query string to narrow the list of items returned. The default is no filter - all resources are returned.	
See the <a href="#">Common Parameters</a> page for more information on the usage of query.		
<b>array of string</b>		
<b>query</b>	A general query string to narrow the list of resources returned. The default is no query - all resources are returned.	string
See the <a href="#">Common Parameters</a> page for more information on the usage of query.		

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

Response Body	EnclosureListV7	
<b>category</b>	Identifies the resource type	string read only
<b>count</b>	The actual number of resources returned in the specified page	integer
<b>created</b>	Date and time when the resource was created	timestamp read only
<b>eTag</b>	Format yyyy-MM-dd'T'HH:mm:ss.SSSZ' 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	string read only
<b>members</b>	List of enclosures resources. array of EnclosureV7	
<b>applianceBayCount</b>	The number of appliance bays in the enclosure.	integer read only
<b>applianceBays</b>	A list of the appliance bays in the enclosure.	Appliance read only
<b>bayNumber</b>	The bay number of the appliance.	integer read only
<b>bayPowerState</b>	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`

The screenshot shows the HPE OneView API Reference interface. On the left, there is a sidebar with a tree view of API categories: Connections, Enclosure Groups (which is selected and highlighted in green), Enclosures, ID Pools, ID Pools IPv4 Ranges, ID Pools IPv4 Subnets, and ID Pools vMAC Ranges. The main area has a search bar at the top with the query 'version'. Below the search bar, a list of API endpoints is displayed, each with a brief description. The first endpoint, 'GET /rest/version', is highlighted with a red box. Other listed endpoints include 'GET /rest/appliance/nodeinfo/version', 'GET /rest/appliance/firmware/notification', 'POST /rest/appliance/eula/save', 'GET /rest/appliance/firmware/pending', and 'POST /rest/appliance/firmware/image'.



You will find something like this:

The screenshot shows the HPE OneView API Reference interface. The left sidebar has a tree view with 'Version' selected. The main content area shows the 'Version' resource details. It describes the version resource as indicating the range of API versions supported by the appliance. A 'GET /rest/version' operation is listed, which returns the range of possible API versions. It notes that the current version is the recommended version to specify in the REST header, while other versions are supported for backward compatibility but might not support the most current features. A 'Get Supported API Versions' section provides instructions for retrieving the minimum and current version of supported REST API versions.

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:

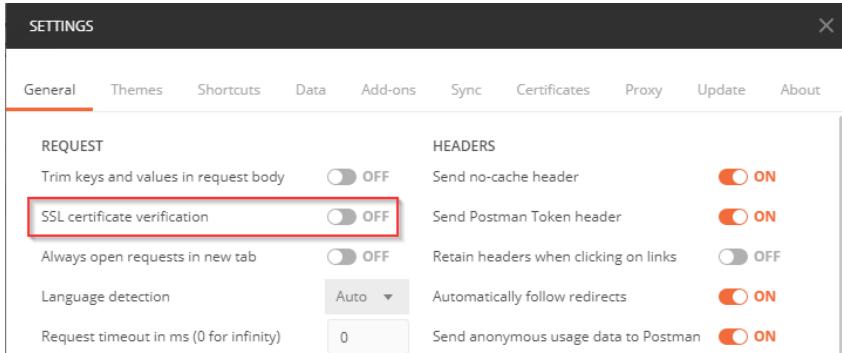
- You can save your REST calls and share the collections with others
- You can use variables to store for instance the OneView authentication session keys

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:

The screenshot shows the Postman interface with a collection named 'OneView' containing 23 requests. A specific POST request titled '1- Login-sessions' is selected. The request details show a POST method to the URL 'https://composer.lj.mougin.net/rest/login-sessions'. The 'Body' tab is active, displaying a JSON response with session information:

```
1 ↵ {  
2 ↵   "sessionID": "NjkyODY4NjA3NDgw1916chT1293kPXOomC913D5YktujFhi2",  
3 ↵   "partnerData": {}  
4 }
```

Postman main pane provides the following:

The screenshot shows the Postman interface with a GET request titled '2- Get-Enclosures' selected. Red arrows point to the following sections:

- Verb:** Points to the 'GET' dropdown in the top left.
- URL:** Points to the URL field 'https://composer.lj.mougin.net/rest/enclosures'.
- Headers:** Points to the 'Headers (2)' tab in the top right.
- Body / Payload:** Points to the 'Body' tab in the top right.

The 'Headers' table contains the following data:

Key	Value	Description
X-API-Version	800	
Auth	<code>{{sessionID}}</code>	
Key	Value	Description



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

The screenshot shows a REST client interface with the following details:

- Top bar: Body, Cookies, Headers (8), Test Results.
- Header area: Status: 200 OK, Time: 48 ms, Size: 59.63 KB, Save, Download.
- Toolbar: Pretty, Raw, Preview, JSON (selected), and a copy icon.
- Main area:
  - Section labeled "Status code" with a red arrow pointing to it.
  - Section labeled "Response" with a red arrow pointing to it.
  - JSON code listing:
- Bottom toolbar: Build, Browse, and several icons.

```
1  "type": "EnclosureListV7",
2  "uri": "/rest/enclosures?start=0&count=40",
3  "category": "enclosures",
4  "eTag": "2018-11-12T15:26:22.251Z",
5  "created": "2018-11-12T15:26:22.251Z",
6  "modified": "2018-11-12T15:26:22.251Z",
7  "start": 0,
8  "count": 3,
9  "total": 3,
10 "prevPageUri": null,
11 "nextPageUri": null,
12 "members": [
13   {
14     "type": "EnclosureV7",
15     "uri": "/rest/enclosures/00000CN7515049L",
16     "category": "enclosures",
17     "eTag": "2018-10-24T10:34:41.720Z",
18     "created": "2018-09-11T11:00:33.315Z",
19     "modified": "2018-10-24T10:34:41.720Z",
20     "refreshState": "NotRefreshing",
21     "stateReason": "None",
22     "enclosureType": "SY12000",
23     "enclosureTypeUri": "/rest/enclosure-types/SY12000",
24     "enclosureModel": "Synergy 12000 Frame",
25     "uuid": "00000CN7515049L",
26     "serialNumber": "CN7515049L",
27     "partNumber": "000000-010",
28     "reconfigurationState": "NotReapplyingConfiguration",
29     "uidState": "Off".
30   }
```

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

The screenshot shows a REST client interface with the following details:

- Request area:
  - Method: GET
  - URI: `https://{{appl}}/rest/version`
  - Header: `Accept-Language: en_US`



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

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

`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`**

The screenshot shows the Postman interface with a red arrow pointing from the text instructions above to the 'Send' button. The 'Send' button is located in the top right corner of the main request configuration area. The URL is set to `https://composer.lj.mougin.net/rest/version`, the method is set to 'GET', and the 'Headers' tab is selected. Under the 'Headers' tab, there is a table with one row. The 'Key' column contains 'Accept' and the 'Value' column contains '`application/json`'. Both the 'Key' and 'Value' cells are circled with red numbers 4 and 5 respectively, corresponding to the numbered steps in the list above.

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.

The screenshot shows the Postman response section. At the top, the status is displayed as 'Status: 200 OK'. Below this, there are tabs for 'Body', 'Cookies', 'Headers', 'Test Results', and 'Save'. The 'Body' tab is selected. Under the 'Body' tab, there are buttons for 'Pretty', 'Raw', 'Preview', and 'JSON'. To the right of the body section, there are 'Save' and 'Download' buttons. The status bar at the bottom of the browser window also shows 'Status: 200 OK', 'Time: 21 ms', and 'Size: 271 B'.



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)

The screenshot shows a browser's developer tools Network tab. The 'Body' tab is selected, displaying a JSON response. The response is:

```
1 {
2     "minimumVersion": 120,
3     "currentVersion": 800
4 }
```

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:

The screenshot shows the HPE OneView API Reference interface. The left sidebar has a tree view with categories like Roles, Security Algorithms, Security Modes, etc., and a 'SEARCH' section. The 'Login Sessions' category is selected and highlighted in green. The main content area has a search bar at the top with the text 'login session'. Below the search bar, the 'Login Sessions' page is displayed. It starts with a brief description: 'Authentication service provides REST APIs to login, reconnect to an existing session, and terminate a session (logout). The login API returns a session token on successful authentication. The reconnect and logout REST APIs require a session token in the request header.' This is followed by several API entries, each with a method (POST, PUT, DELETE), a URL path, and a brief description:

- POST /rest/login-sessions**: Authenticate user with specified credentials. User name, password and an optional directory are specified as input in the request body. X-Api-Version to be provided in the header.
- PUT /rest/login-sessions**: Reconnect to an existing session that is neither explicitly logged out nor timed out. Session token must be provided in the request header.
- DELETE /rest/login-sessions**: Remove user session by invoking an explicit logout. Session token must be provided in the request header.
- POST /rest/login-sessions/auth-token**: Creates a new user session from an existing session, controlling which user assigned permissions are active and inactive.
- POST /rest/login-sessions/smartcards**: Authenticate Enterprise Directory user logging in with certificate and X-Api-Version to be provided in the header. The user is required to use a client authentication in addition to the normal server authentication during TLS negotiation. Here the private key will be used by the client library.

Ellipsis (...)



If you expand the first POST, you have:

▼ POST /rest/login-sessions

Authenticate user with specified credentials. User name, password and an optional directory are specified as input in the request body. X-Api-Version to be provided in the header.

**Authenticate User**

Authenticate user - administrator from directory - mydirectory.

Request

```
POST https://{{appl}}/rest/login-sessions

X-Api-Version: 800
Content-Type: application/json ← Header

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

The diagram shows a POST request to the URI /rest/login-sessions. It includes two HTTP headers: X-Api-Version: 800 and Content-Type: application/json. The request body is a JSON object with four fields: authLoginDomain, password, userName, and loginMsgAck. Red arrows point from the text labels 'Header' and 'Body / Payload' to their respective parts of the request message.

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	<code>https://{{appl}}/rest/login-sessions</code>
Verb	<code>POST</code>
Header	<code>Content-type:application/json</code> <code>X-API-Version:800</code>
Payload	<pre>{   "authLoginDomain": "mydirectory",   "password": "mypassword",   "userName": "administrator",   "loginMsgAck": "true" }</pre>

### Content of the headers:

The screenshot shows the Postman interface with the 'Headers' tab selected. It contains two entries: 'Content-Type' with value 'application/json' and 'x-api-version' with value '800'. Other tabs like 'Params', 'Authorization', 'Body', 'Pre-request Script', 'Tests', 'Cookies', and 'Code' are also visible.

KEY	VALUE	DESCRIPTION	... Bulk Edit Presets
Content-Type	application/json		
x-api-version	800		
Key	Value	Description	

### Content of the Body:

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

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

The screenshot shows the Postman interface with the 'Body' tab selected (marked with a red circle 1). Under 'Content Type', 'raw' is selected (marked with a red circle 2). The payload area (marked with a red circle 3) contains the JSON code from the previous step.

Then press **Send**.



The response we get is the following:

POST https://composer.lj.mougin.net/rest/login-sessions

Body (JSON)

```
1 {  
2   "password": "password",  
3   "userName": "administrator"  
4 }  
5
```

Status: 200 OK Time: 49 ms Size: 309 B

Response

```
1 {  
2   "sessionID": "LTMS00KzNjE0MTQyxfU_eC1wFGp_B1aEGaG1NM_e89DN8Lv2",  
3   "partnerData": {}  
4 }
```

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	<code>https://composer.1j.lab/rest/global-settings</code>
Verb	<code>GET</code>
Header	<code>Auth : &lt;your sessionID&gt;</code> <code>X-API-Version:800</code>



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

The screenshot shows the Postman interface with a GET request to `https://composer.lj.mougin.net/rest/global-settings`. The Headers tab is active, displaying two entries: `x-api-version` with value `800` and `Auth` with value `LTM5ODkzNjE0MTQyxfU_eCIWFGp_BlaEGaGINM...`. A red arrow points from the text "SessionID" to the yellow-highlighted `Auth` header value.

Then when all set, press **Send**

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

The screenshot shows the Postman interface with the response body displayed in JSON format. The response is a paginated collection of global settings, with the first few members shown:

```
1  {
2   "type": "SettingsPaginatedCollectionV2",
3   "uri": "/rest/global-settings?start=0&count=50",
4   "category": "global-settings",
5   "eTag": null,
6   "created": null,
7   "modified": null,
8   "start": 0,
9   "count": 26,
10  "total": 26,
11  "prevPageUri": null,
12  "nextPageUri": null,
13  "members": [
14    {
15      "type": "SettingV2",
16      "uri": "/rest/global-settings/appliance/global/alertMax",
17      "category": "global-settings",
18      "eTag": null,
19      "created": "2018-09-11T08:55:35.652Z",
20      "modified": "2018-09-11T08:55:35.652Z",
21      "name": "alertMax",
22      "value": "75000",
23      "group": "global",
24      "settingCategory": "appliance",
25      "description": null,
26      "state": null,
27      "status": null
28    },
29    {
30      "type": "SettingV2",
31      "uri": "/rest/global-settings/appliance/global/alertMaxDeviation",
32      "category": "global-settings",
33      "eTag": null,
34      "created": "2018-09-11T08:55:35.875Z",
35      "modified": "2018-09-11T08:55:35.875Z",
36      "name": "alertMaxDeviation",
37      "value": "800",
38      "group": "global",
39      "settingCategory": "appliance",
40      "description": null,
41      "state": null,
42      "status": null
43    }
44  ]}
```



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

The screenshot shows the Postman interface with the 'Headers' tab selected. There are two headers listed: 'x-api-version' with value '800' and 'Auth' with value 'LTM5ODkzNjE0MTQyxfU\_eCiWFGp\_BlaEGaGINM...'. The 'Auth' header is highlighted with a red border. Below the headers, the 'Body' tab is selected, showing a JSON response:

```
1 <pre>{
2   "errorCode": "AUTHORIZATION_MISSING_AUTH_HEADER",
3   "message": "Authorization error: Missing 'auth' header.",
4   "details": "Missing 'auth' header from the request.",
5   "recommendedActions": [
6     "Please provide the missing 'auth' header value and try again."
7   ],
8   "errorSource": null,
9   "nestedErrors": [],
10  "data": {}
11 }</pre>
```

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**

HPE OneView					
API Reference					
About	SERVERS	NETWORKING	STORAGE	FC-SANS	FACILITIES
Security Model	Connections	Connection Templates	Drive Enclosures	Endpoints	Datacenters
Response Codes	Enclosure Groups	Ethernet Networks	SAS Logical JBOD Attachments	Managed SANs	Power Devices
Association Names	Enclosures	Fabric Managers	SAS Logical JBODs	Providers	Racks
Common Parameters	ID Pools	Fabrics	Storage Pools	SAN Managers	Unmanaged Devices
Common Attributes	ID Pools IPv4 Ranges	FC Networks	Storage Systems		
What's New?	ID Pools IPv4 Subnets	FCoE Networks	Storage Templates		
	ID Pools vMAC				

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	<code>https://composer.1j.lab/rest/ethernet-networks</code>
Verb	<code>GET</code>
Header	<code>Auth : your sessionID</code> <code>X-API-Version:800</code>



The body response is the following:

GET https://composer.lj.mougin.net/rest/ethernet-networks

Headers (2)

Key	Value	Description
X-API-Version	800	
Auth	MzAxODYyNzM2MzU2sjnl5g50ivap3b5U45CkIos...	
Key	Value	Description

Body (Pretty) Raw Preview JSON

```
1  {
2      "type": "NetworkCollectionV4",
3      "uri": "/rest/ethernet-networks?start=0&count=9",
4      "category": "ethernet-networks",
5      "eTag": null,
6      "created": null,
7      "modified": null,
8      "start": 0,
9      "count": 9,
10     "total": 9,
11     "prevPageUri": null,
12     "nextPageUri": null,
13     "members": [
14         {
15             "type": "ethernet-networkV4",
16             "uri": "/rest/ethernet-networks/0770ecc1-dfbd-4173-9e8d-be563f518001",
17             "category": "ethernet-networks",
18             "eTag": "205ac1d5-0587-4916-a45d-d45c224685e6",
19             "created": "2018-09-26T15:23:51.624Z",
20             "modified": "2018-09-26T15:23:51.625Z",
21             "scopesUri": "/rest/scopes/resources/rest/ethernet-networks/0770ecc1-dfbd-4173-9e8d-be563f518001",
22             "vlanId": 6,
23             "subnetUri": null,
24             "connectionTemplateUri": "/rest/connection-templates/887a905d-cb3f-4d8b-ad66-638fc4154a04",
25             "privateNetwork": false,
26             "smartLink": false,
27             "purpose": "FaultTolerance",
28             "ethernetNetworkType": "Tagged",
29             "fabricUri": "/rest/fabrics/f54fcba6-59cf-4d4a-b6cc-517ccb17f6df",
30             "description": null,
31             "state": "Active",
32             "status": "OK",
33             "name": "Cluster-Network"
34         }
35     ]
36 }
```

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:

Params Authorization Headers (2) Body Pre-request Script Tests ● Cookies Code

KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
X-API-Version	300				
Auth	MzAxODYyNzM2MzU2sjnI5gS0ivap3b5U45CkIOs...				
Key	Value	Description			

Body Cookies Headers (7) Test Results Status: 200 OK Time: 30 ms Size: 5.9 KB Save Download

Pretty Raw Preview JSON ▾  

```
1 ▾ {  
2     "type": "NetworkCollectionV300",  
3     "uri": "/rest/ethernet-networks?start=0&count=9",  
4     "category": "ethernet-networks",  
5     "eTag": null,  
6     "created": null,  
7     "modified": null,  
8     "start": 0,  
9     "count": 9,  
10    "total": 9,  
11    "prevPageUri": null,  
12    "nextPageUri": null,  
13    "members": [  
14        {  
15            "type": "ethernet-networkV300",  
16            "uri": "/rest/ethernet-networks/0770ecc1-dfb4-4173-9e8d-be563f518001",  
17            "category": "ethernet-networks",  
18            "eTag": "205ac1d5-0587-4916-a45d-d45c224685e6",  
19            "created": "2018-09-26T15:23:51.624Z",  
20            "modified": "2018-09-26T15:23:51.625Z",  
21            "vlanId": 6,  
22            "subnetUri": null,  
23            "connectionTemplateUri": "/rest/connection-templates/887a905d-cb3f-4d8b-ad66-638fc4154a04",  
24            "privateNetwork": false,  
25            "smartlink": false,  
26            "purpose": "FaultTolerance",  
27            "ethernetNetworkType": "Tagged",  
28            "fabricUri": "/rest/fabrics/f54fcb46-59cf-4d4a-b6cc-517ccb17f6df",  
29            "description": null,  
30            "state": "Active",  
31            "status": "OK",  
32            "name": "Cluster-Network"  
33        },  
34    ]  
35},  
36  
37
```

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/f54fcbb46-59cf-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

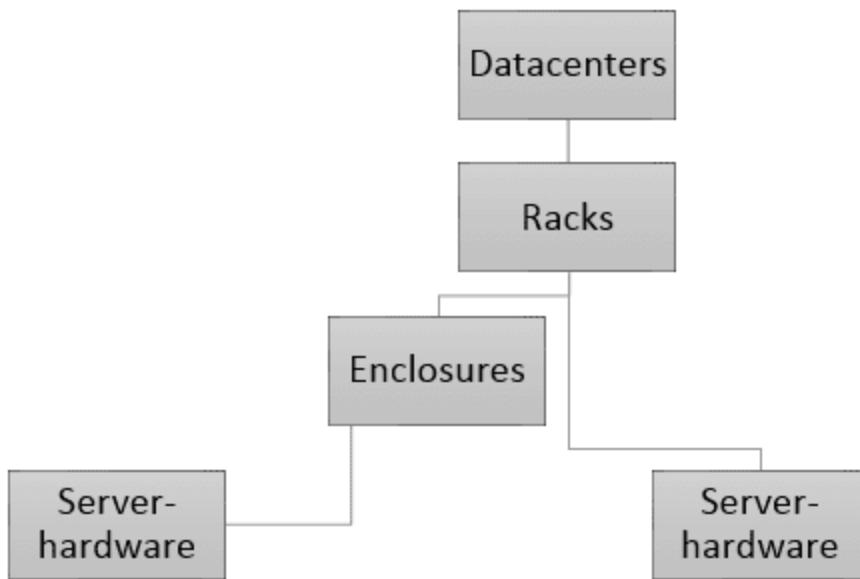
In previous labs, we saw some of the concept on the versioning, the authentication and the discovery of the infrastructure, via a REST API approach.

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.

The screenshot shows the Postman interface with a GET request to <https://composer.lj.mougin.net/rest/datacenters/schema>. The Headers tab is active, displaying the following key-value pairs:

KEY	VALUE	DESCRIPTION
X-API-Version	400	
Auth	MzAxODYyNzM2MzU2sjnl5gS0ivap3b5U45CkIOs...	
Key	Value	Description

The Body tab shows the JSON response from the API, which defines a schema for a data center object:

```
1 <object>
2   "type": "object",
3   "properties": {
4     "width": {
5       "type": "integer",
6       "description": "The width in millimeters of the data center. This is associated with the PhysicalAllocation x position .",
7       "maximum": "50000",
8       "minimum": "1000",
9       "required": "true"
10    },
11    "depth": {
12      "type": "integer",
13      "description": "The depth in millimeters of the data center. This is associated with the PhysicalAllocation y position .",
14      "maximum": "50000",
15      "minimum": "1000",
16      "required": "true"
17    },
18    "coolingCapacity": {
19      "type": "integer",
20      "description": "Maximum cooling capacity for the datacenter in watts.",
21      "minimum": "0"
22    },
23    "costPerKilowattHour": {
24      "type": "number",
25      "description": "The energy cost per kilowatt-hour."
26    },
27    "currency": {
28      "type": "string",
29      "description": "The currency unit for energy costs.",
30      "required": "true",
31      "default": "USD"
32    },
33    "deratingType": {
34      "type": "string"
35  }
```

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.

The screenshot shows a JSON response from a REST API. The status bar at the top indicates "Status: 200 OK Time: 64 ms Size: 3.97 KB". Below the status bar, there are tabs for "Body", "Cookies", "Headers (7)", and "Test Results". The "JSON" tab is selected. The main content area displays the JSON schema for a Datacenter object. The schema includes properties like width, depth, coolingCapacity, costPerKilowatthour, currency, deratingType, deratingPercentage, coolingMultiplier, contents, category, eTag, created, modified, id, uid, defaultPowerLineVoltage, name, url, state, and status. A detailed description is provided for the Datacenter object: "A data center object represents a facility used to house computer systems and associated components, such as telecommunications and storage systems. Generally represents a contiguous area of racks containing IT equipment." The code block is numbered from 1 to 130.

```
1 ▾ [{}  
2   "type": "object",  
3   "properties": {  
4     "width": {},  
5     "depth": {},  
6     "coolingCapacity": {},  
7     "costPerKilowatthour": {},  
8     "currency": {},  
9     "deratingType": {},  
10    "deratingPercentage": {},  
11    "coolingMultiplier": {},  
12    "contents": {},  
13    "category": {},  
14    "eTag": {},  
15    "created": {},  
16    "modified": {},  
17    "id": {},  
18    "uid": {},  
19    "defaultPowerLineVoltage": {},  
20    "name": {},  
21    "url": {},  
22    "state": {},  
23    "status": {}  
24  },  
25  "description": "A data center object represents a facility used to house computer systems and associated components, such as telecommunications and storage systems. Generally represents a contiguous area of racks containing IT equipment."  
130 }]
```

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

The screenshot shows a JSON response from a REST API. The schema for a Datacenter object is shown, with the "width" property expanded. The "width" property is defined as an integer with a description: "The width in millimeters of the data center. This is associated with the PhysicalLocation x position.". It also specifies a maximum value of 50000 and a minimum value of 1000, and is marked as required. The code block is numbered from 1 to 11.

```
1 ▾ [{}  
2   "type": "object",  
3   "properties": {  
4     "width": {  
5       "type": "integer",  
6       "description": "The width in millimeters of the data center. This is associated with the PhysicalLocation x position.",  
7       "maximum": "50000",  
8       "minimum": "1000",  
9       "required": "true"  
10     },  
11     "depth": {},  
12   },  
13   "deratingType": {},  
14 }
```

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*.

The screenshot shows a JSON response from a REST API. The schema for a Datacenter object is shown, with the "currency" property expanded. The "currency" property is defined as a string with a description: "The currency unit for energy costs.", it is marked as required, and the default value is USD. The code block is numbered from 27 to 33.

```
27 ▾ [{}  
28   "currency": {  
29     "type": "string",  
30     "description": "The currency unit for energy costs.",  
31     "required": "true",  
32     "default": "USD"  
33   },  
34   "deratingType": {},  
35 }
```



*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": { },
51 "contents": {
52     "type": "array",
53     "items": {
54         "type": "object",
55         "properties": {
56             "x": {
57                 "type": "number",
58                 "description": "The coordinate of the front left corner of the unrotated
59                     resource in the data center layout width axis where the given resource is
60                     located. Units are in millimeters.",
61                 "required": "true"
62             },
63             "y": {
64                 "type": "number",
65                 "description": "The coordinate of the front left corner of the unrotated
66                     resource in the data center layout depth axis where the given resource is
67                     located. Units are in millimeters.",
68                 "required": "true"
69             },
70             "rotation": {
71                 "type": "number",
72                 "description": "The rotation (degrees) from 0-359 around the center of the
73                     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         "description": "The collection of physical resources (racks) in the data center and their
82             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:

**HPE OneView API Reference**

**Search**

**Response Body**

Datacenter List			
SAN Managers	<b>category</b>	Identifies the resource type	string read only
FACILITIES	<b>count</b>	The actual number of resources returned in the specified page	integer
Datacenters	<b>created</b>	Date and time when the resource was created	timestamp read only
Power Devices	<b>eTag</b>	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
Racks	<b>members</b>	The array of resources contained in the specified collection	
Unmanaged Devices	<b>array of Datacenter</b>		
EXTERNAL MANAGERS	<b>category</b>	Identifies the resource type	string read only
Hypervisor Managers	<b>contents</b>	The collection of physical resources (racks) in the data center and their position.	
HYPERVERISORS	<b>resourceUri</b>	The uri of the rack resource whose position is described.	string
Hypervisor Cluster Profiles	<b>rotation</b>	The rotation (degrees) from 0-359 around the center of the resource.	decimal
Hypervisor Host Profiles	<b>x</b>	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.	decimal required
Metric Streaming	<b>y</b>	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.	decimal required
Remote Syslog			
ACTIVITY			
Alerts			
Audit Logs			
Events			
Reports			
Tasks			
SETTINGS			
Appliance Configuration Timeconfig			
Appliance Data Collection			
Appliance Device Read Community String			



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>

GET https://composer.lj.mougin.net/rest/datacenters Send Save

Params Authorization Headers (2) Body Pre-request Script Tests Cookies Code

	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	x-api-version	800				
<input checked="" type="checkbox"/>	Auth	LTUxNjQzODcxMDU44b3WAhOGGxVh...				
	Key	Value	Description			

Body Cookies Headers (7) Test Results Status: 200 OK Time: 30 ms Size: 1.21 KB Download

Pretty Raw Preview JSON  

```
1 [ { 2   "type": "ErmResourceCollection", 3   "uri": "/rest/datacenters?start=0&count=1000", 4   "category": "datacenters", 5   "eTag": null, 6   "created": "2018-11-14T14:42:00.606Z", 7   "modified": "2018-11-14T14:42:00.606Z", 8   "start": 0, 9   "count": 1, 10  "total": 1, 11  "prevPageUri": null, 12  "nextPageUri": null, 13  "members": [ 14    { 15      "width": 10668, 16      "depth": 13716, 17      "coolingCapacity": 350, 18      "costPerKilowattHour": 0.1, 19      "currency": "USD", 20      "deratingType": "NaJp", 21      "deratingPercentage": 20, 22      "coolingMultiplier": 1.5, 23      "contents": [ 24        { 25          "x": 1000, 26          "y": 1200, 27          "rotation": 0, 28          "resourceUri": "/rest/racks/e7a03508-f1b6-4ec9-903f-1edc7107b26e" 29        } 30      ], 31      "remoteSupportUri": "/rest/support/datacenters/d772ebb8-e7d6-4734-b1a4-6ced162f322e", 32      "category": "datacenters", 33      "eTag": "2018-09-11T11:47:18.579Z", 34      "created": "2018-09-11T11:47:00.684Z", 35      "modified": "2018-09-11T11:47:18.579Z", 36      "id": "d772ebb8-e7d6-4734-b1a4-6ced162f322e", 37      "uuid": "d772ebb8-e7d6-4734-b1a4-6ced162f322e", 38      "defaultPowerLineVoltage": 220, 39      "name": "Sophia-Antipolis", 40      "uri": "/rest/datacenters/d772ebb8-e7d6-4734-b1a4-6ced162f322e", 41      "state": "Unmanaged" 42    } 43  ] }
```

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 URL: /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.

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: https://composer.lj.mougin.net/rest/racks/schema
- Headers tab selected, showing:
  - x-api-version: 800
  - Auth: LTUxNjQzODcxMDU44b3WApOGGxVh...
  - Key: Value
- Body tab selected, showing the JSON schema for a rack:

```
1 [ {  
2   "type": "object",  
3   "properties": {  
4     "uHeight": { },  
5     "width": { },  
6     "height": { },  
7     "depth": { },  
8     "thermallimit": { },  
9     "partNumber": { },  
10    "serialNumber": { },  
11    "rackMounts": { },  
12    "category": { },  
13    "eTag": { },  
14    "created": { },  
15    "modified": { },  
16    "id": { },  
17    "uuid": { },  
18    "name": { },  
19    "uri": { },  
20    "state": { },  
21    "model": { },  
22    "status": { }  
23  },  
24  "description": "A rack is a physical structure that contains IT equipment such as enclosures, servers, power delivery devices and unmanaged devices, in a data center."  
25 } ]
```

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.

```
1 [ {
2   "type": "object",
3   "properties": {
4     "uHeight": { },
5     "width": { },
6     "height": { },
7     "depth": { },
8     "thermalLimit": { },
9     "partNumber": { },
10    "serialNumber": { },
11    "rackMounts": {
12      "type": "array",
13      "items": {
14        "type": "object",
15        "properties": {
16          "mountUri": { },
17          "topUSlot": { },
18          "uHeight": { },
19          "relativeOrder": { },
20          "location": { }
21        }
22      }
23    },
24    "description": "References to the resources contained in the rack."
25  },
26  "category": { },
27  "eTag": { },
28  "created": { },
29  "modified": { },
30  "id": { },
31  "uuid": { },
32  "name": { },
33  "uri": { },
34  "state": { },
35  "model": { },
36  "status": { }
37 },
38 "description": "A rack is a physical structure that contains IT equipment such as enclosures, servers, power delivery devices and unmanaged devices, in a data center."
39 }
```



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

```
1  {
2      "type": "ErmResourceCollection",
3      "uri": "/rest/racks?start=0&count=1000",
4      "category": "racks",
5      "eTag": null,
6      "created": "2018-11-14T15:54:14.930Z",
7      "modified": "2018-11-14T15:54:14.930Z",
8      "start": 0,
9      "count": 1,
10     "total": 1,
11     "prevPageUri": null,
12     "nextPageUri": null,
13     "members": [
14         {
15             "uHeight": 36,
16             "width": 600,
17             "height": 2032,
18             "depth": 1075,
19             "thermalLimit": 10000,
20             "partNumber": "AF046A",
21             "serialNumber": "AA8B1122CCDD",
22             "packMounts": [
23                 {
24                     "mountUri": "/rest/enclosures/000000CN7515049L",
25                     "topUSlot": 20,
26                     "uHeight": 10,
27                     "relativeOrder": -1,
28                     "location": "CenterFront"
29                 },
23                 {
30                     "mountUri": "/rest/enclosures/000000CN7515049C",
31                     "topUSlot": 10,
32                     "uHeight": 10,
33                     "relativeOrder": -1,
34                     "location": "CenterFront"
35                 },
36                 {
37                     "mountUri": "/rest/enclosures/000000CN7516060D",
38                     "topUSlot": 30,
39                     "uHeight": 10,
40                     "relativeOrder": -1,
41                     "location": "CenterFront"
42                 }
43             ],
44             "model": "",
45             "category": "racks",
46             "eTag": "2018-11-14T15:54:08.517Z",
47             "created": "2018-09-11T11:47:16.079Z",
48             "modified": "2018-11-14T15:54:08.517Z",
49             "id": "e7a03508-f1b6-4ec9-903f-1edc7107b26e",
50             "uuid": "e7a03508-f1b6-4ec9-903f-1edc7107b26e",
51             "name": "Synergy-Rack",
52             "uri": "/rest/racks/e7a03508-f1b6-4ec9-903f-1edc7107b26e",
53             "state": "Unmanaged",
54             "status": "OK"
55         }
56     ]
57 }
```

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:

```
13  "members": [
14    {
15      "uHeight": 36,
16      "width": 600,
17      "height": 2032,
18      "depth": 1075,
19      "thermalLimit": 10000,
20      "partNumber": "AF046A",
21      "serialNumber": "AABB1122CCDD",
22      "rackMounts": [
23        {
24          "mountUri": "/rest/enclosures/000000CN7515049L",
25          "topUSlot": 20, 
26          "uHeight": 10,
27          "relativeOrder": -1,
28          "location": "CenterFront"
29        },
      
```

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:

The screenshot shows the Postman Settings interface. The top bar has a close button (X). Below it, the 'General' tab is selected. The 'REQUEST' section contains options like 'Trim keys and values in request body' (OFF), 'SSL certificate verification' (OFF), 'Always open requests in new tab' (OFF), 'Language detection' (Auto), and 'Request timeout in ms (0 for infinity)' (0). The 'HEADERS' section contains several options, all of which are currently turned ON: 'Send no-cache header', 'Send Postman Token header', 'Retain headers when clicking on links' (which is highlighted with a red box), 'Automatically follow redirects', and 'Send anonymous usage data to Postman'.

```
1  "type": "EnclosureV7",
2  "uri": "/rest/enclosures/00000CN7515049L",
3  "category": "enclosures",
4  "eTag": "2018-10-24T10:34:41.720Z",
5  "created": "2018-09-11T11:00:33.315Z",
6  "modified": "2018-10-24T10:34:41.720Z",
7  "refreshState": "NotRefreshing",
8  "stateReason": "None",
9  "enclosureType": "SY12000",
10 "enclosureTypeUri": "/rest/enclosure-types/SY12000",
11 "enclosureModel": "Synergy 12000 Frame",
12 "uuid": "00000CN7515049L",
13 "serialNumber": "CN7515049L",
14 "partNumber": "00000-010",
15 "reconfigurationState": "NotReapplyingConfiguration",
16 "uidState": "Off",
17 "licensingIntent": "NotApplicable",
18 "deviceBayCount": 12,
19 "deviceBays": [ ],
20 "interconnectBayCount": 6,
21 "interconnectBays": [ ],
22 "fanBayCount": 10,
23 "fanBays": [ ],
24 "powerSupplyBayCount": 6,
25 "powerSupplyBays": [ ],
26 "enclosureGroupUri": "/rest/enclosure-groups/d8b4553b-dfb0-4ec7-b58e-97aeea8e776b",
27 "fwBaselineUri": "",
28 "fwBaselineName": "",
29 "isFwManaged": true,
30 "forceInstallFirmware": false,
31 "logicalEnclosureUri": "/rest/logical-enclosures/69d41e58-a12d-44a7-9dcc-0e6bc938fcde",
32 "managerBays": [ ],
33 "supportState": "Disabled",
34 "supportDataCollectionState": null,
35 "supportDataCollectionType": null,
36 "supportDataCollectionsUri": "/rest/support/data-collections?deviceID=00000CN7515049L&category=enc",
37 "remoteSupportUri": "/rest/support/enclosures/00000CN7515049L",
38 "remoteSupportSettings": { },
39 "crossBars": [],
40 "partitions": [],
41 "scopesUri": "/rest/scopes/resources/rest/enclosures/00000CN7515049L",
42 "status": "OK",
43 "name": "Frame2-CN7515049L",
44 "state": "Configured",
45 "description": null,
46 "frameLinkModuleDomain": "local",
47 "version": "G1",
48 "powerMode": "RedundantPowerFeed",
49 "managerBayCount": 2,
50 "fansAndManagementDevicesWatts": 690,
51 "powerCapacityBoostWatts": 0,
52 "minimumPowerSupplies": 1,
53 "minimumPowerSuppliesForRedundantPowerFeed": 2,
54 "powerAvailableWatts": 6459,
```

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.

```

18 "licensingIntent": "NotApplicable",
19 "deviceBayCount": 12,
20 "deviceBays": [
21 {
22     "type": "DeviceBayV400",
23     "bayNumber": 1,
24     "model": null,
25     "devicePresence": "Present",
26     "profileUri": "/rest/server-profiles/2a2b4d28-c59a-4455-9996-b2e1bceb31a6",
27     "deviceUri": "/rest/server-hardware/36343537-3338-4E43-3736-303130423734",
28     "coveredByProfile": "/rest/server-profiles/2a2b4d28-c59a-4455-9996-b2e1bceb31a6",
29     "coveredByDevice": "/rest/server-hardware/36343537-3338-4E43-3736-303130423734",
30     "ipv4Setting": {
31         "ipAddress": null,
32         "mode": "DHCP",
33         "ipAssignmentState": "None",
34         "ipRangeUri": null
35     },
36     "uri": "/rest/enclosures/000000CN7515049C/device-bays/1",
37     "category": "device-bays",
38     "eTag": null,
39     "created": null,
40     "modified": null,
41     "availableForHalfHeightProfile": false,
42     "availableForFullHeightProfile": false,
43     "deviceBayType": "SY12000DeviceBay",
44     "deviceFormFactor": "SingleHeightSingleWide",
45     "bayPowerState": "Unknown",
46     "changeState": "None",
47     "availableForHalfHeightDoubleWideProfile": false,
48     "availableForFullHeightDoubleWideProfile": false,
49     "uuid": null,
50     "powerAllocationWatts": 0,
51     "serialConsole": true
52 },
53 },
54 {
55     "type": "DeviceBayV400",
56     "bayNumber": 2,
57     "model": null,
58     "devicePresence": "Absent",
59     "profileUri": null,
60     "deviceUri": null,
61     "coveredByProfile": null,
62     "coveredByDevice": null,
63     "ipv4Setting": null,
64     "uri": "/rest/enclosures/000000CN7515049C/device-bays/2",
65     "category": "device-bays",
66     "eTag": null,
67     "created": null,
68     "modified": null,
69     "availableForHalfHeightProfile": true,
70     "availableForFullHeightProfile": false,
71     "deviceBayType": "SY12000DeviceBay",
72     "deviceFormFactor": "Empty",
73     "bayPowerState": "Unknown",
74     "changeState": "None",
75     "availableForHalfHeightDoubleWideProfile": false,
76     "availableForFullHeightDoubleWideProfile": false,
77     "uuid": null,
78     "nowAllocationWatts": 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.

```
1  {
2    "type": "ServerProfileTemplateListV5",
3    "uri": "/rest/server-profile-templates?start=0&count=64",
4    "category": "server-profile-templates",
5    "eTag": "2018-11-19T16:24:38.935Z",
6    "created": "2018-11-19T16:24:38.935Z",
7    "modified": "2018-11-19T16:24:38.935Z",
8    "start": 0,
9    "count": 7,
10   "total": 7,
11   "prevPageUri": null,
12   "nextPageUri": null,
13   "members": [
14     {
15       "type": "ServerProfileTemplateV5",
16       "uri": "/rest/server-profile-templates/02d63533-5e81-41eb-88c7
-861d38d6ea7a",
17       "name": "ESXi 6.5U1 deployment with Streamers",
18       "description": "",
19       "serverProfileDescription": "",
20       "serverHardwareTypeUri": "/rest/server-hardware-types/E282CA4E-13D8-441B
-B065-E66F284D0216",
21       "enclosureGroupUri": "/rest/enclosure-groups/d8b4553b-dfb0-4ec7-b58e
-97aeaa8e776b",
22       "affinity": "Bay",
23       "hideUnusedFlexNics": true,
24       "macType": "Virtual",
25       "wwnType": "Virtual",
26       "serialNumberType": "Virtual",
27       "iscsiInitiatorNameType": "AutoGenerated",
28       "osDeploymentSettings": {
29         "osDeploymentPlanUri": "/rest/os-deployment-plans/27730614-ae4f-4828
-8ca3-45cc56b2db22",
30         "osCustomAttributes": [
31           {
32             "name": "ManagementNIC.dhcp",
33             "value": "false",
34             "constraints": null,
35           }
36         ]
37       }
38     }
39   ]
40 }
```



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

The screenshot shows the HPE OneView interface with the 'Server Profile Templates' tab selected. The left sidebar lists several templates, with 'ESXi deployment with Streamers' highlighted. The main content area displays the details for this template, including its general configuration and associated server profiles. A green box highlights the 'Server Profiles' link in the top right corner of the main content area.

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:

The screenshot shows the HPE OneView API Reference page with the 'Association Names' section selected. The left sidebar lists various resource types. The main content area shows a table of associations between 'Parent' resources and 'Child' resources, with a column labeled 'Association name'. A green box highlights the 'Association name' column in the table.

Parent	Child	Association name
connections	interconnects	CONNECTION_TO_INTERCONNECT
datacenters	racks	DATACENTER_TO_PHYSICAL_OBJECT
drive-enclosures/drive-bays	drive-enclosures/drive-bays/drive	DRIVE_BAY_TO_DRIVE_ASSOC
enclosures	enclosures/device-bays	ENCLOSURE_TO_DEVICE_BAY
	interconnects	ENCLOSURE_TO_INTERCONNECT
	logical-interconnects	ENCLOSURE_TO_LOGICAL_INTERCONNECT
	server-hardware	ENCLOSURE_TO_BLADE
enclosure-groups	enclosures	ENCLOSURE_GROUP_TO_ENCLOSURE
	logical-enclosures	ENCLOSURE_GROUP_TO_LOGICAL_ENCLOSURE
	logical-interconnect-groups	ENCLOSURE_GROUP_TO_LOGICAL_INTERCONNECT_GROUP
	server-profiles	enclosure_group_to_server_profiles
	server-profile-templates	ENCLOSURE_GROUP_TO_SERVER_PROFILE_TEMPLATE
enclosure-types	enclosures	ENCLOSURE_TYPE_TO_ENCLOSURE
enclosures/device-bays	server-hardware	DEVICE_BAY_TO_SERVER_HARDWARE
fc-device-managers	server-profiles	DEVICE_BAY_TO_PROFILE
fc-sans	fc-networks	SAN_MANAGER_TO_MANAGED_SAN
fc-networks		MANAGED_DA_SAN_TO_FC_NETWORK



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`

The screenshot shows the HPE OneView API Reference interface. On the left, a sidebar lists various categories like About, Security Model, Response Codes, Association Names (which is selected and highlighted in green), Common Parameters, Common Attributes, What's New?, SERVERS, and Connections. The main content area has a search bar at the top. Below it, under the 'Association Names' section, there is a tree view. The root node is 'server-profile-templates'. Underneath it are three groups: 'ethernet-networks', 'fc-networks', and 'storage-pools'. 'ethernet-networks' contains 'storage-volumes' and 'storage-volume-templates'. 'fc-networks' contains 'storage-volumes' and 'storage-volume-templates'. 'storage-pools' contains 'storage-volumes' and 'storage-volume-templates'. A red arrow points from the 'server-profile' node under 'fc-networks' to the text 'server\_profile\_template\_to\_server\_profiles' which is highlighted in yellow.

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**.

The screenshot shows the 'Association names' page of the HPE OneView API Reference. The left sidebar is identical to the previous screenshot. The main content area has a heading 'Association names'. Below it, a text block says: 'HPE OneView provides a fully connected resource model. The following associations describe the relationships between the HPE OneView resources. See the [associations API](#) documentation for details on accessing and using associations.' A red box highlights the 'associations API' link. Below this, there is a table with columns: Parent, Child, and Association name. The table lists various associations:

Parent	Child	Association name
connections	interconnects	CONNECTION_TO_
datacenters	racks	DATACENTER_TO_
drive-enclosures/drive-bays	drive-enclosures/drive-bays/drive	DRIVE_BAY_TO_D
enclosures	enclosures/device-bays	ENCLOSURE_TO_D
	interconnects	ENCLOSURE_TO_I
	logical-interconnects	ENCLOSURE_TO_L
	server-hardware	ENCLOSURE_TO_B
enclosure-groups	enclosures	ENCLOSURE_GROU
	logical-enclosures	ENCLOSURE_GROU
	logical-interconnect-groups	ENCLOSURE_GROU



The screenshot shows the HPE OneView API Reference interface. The left sidebar has a tree view with categories like HPE OneView, API Reference, Roles, Security Algorithms, Security Modes, Security Protocols, Server Certificates, Sessions, SSHHostKeys, Standards, Compatibility, User Settings, and Users. Under SEARCH, 'Index Associations' is selected. The main content area has a search bar and a title 'Index Associations'. Below it, a note says 'Manage parent/child associations between resources. Use these APIs to create, name, delete, and search associations, generating association schema in JSON format, and so on.' A section titled 'Details and syntax of Index specific query parameters' contains examples of URLs with filter parameters. It includes examples for case insensitive and sensitive searches, and a null check.

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://{{app1}}/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	<code>https://composer.lj.lab/rest/server-profile-templates</code>
Params Key	<code>filter</code>
Params Value	<code>"name matches 'Template for Prod-ESX servers'"</code>
Verb	<code>GET</code>
Header	<code>Auth : your sessionID</code> <code>X-API-Version:800</code>

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 screenshot shows the Postman interface with a GET request to the specified URL. The 'Params' tab is selected, showing a single parameter named 'filter' with the value 'name matches 'Template for Prod-ESX servers''. The 'Body' tab shows the JSON response from the server.

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

The screenshot shows the Postman interface displaying the JSON response body. The response is a list of server profile templates, with one specific template highlighted in yellow: "Template for Prod-ESX servers".

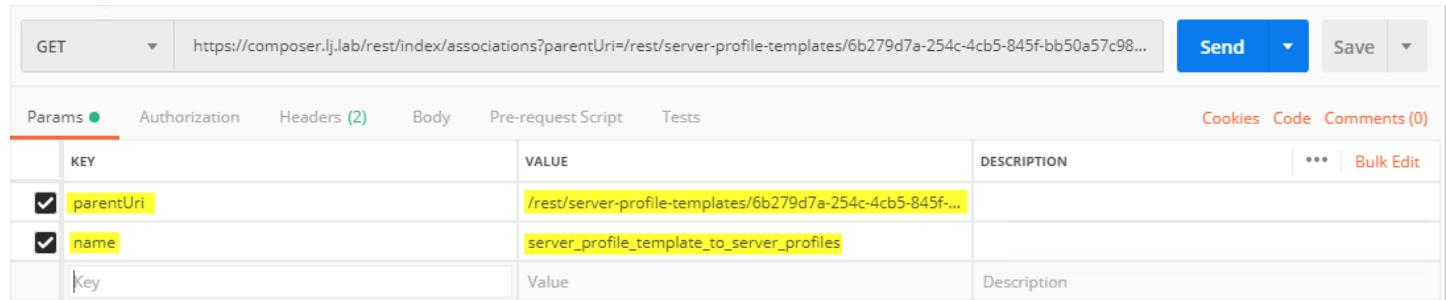
```
1. {
2.     "type": "ServerProfileTemplateListV5",
3.     "uri": "/rest/server-profile-templates?filter=%22name%20matches%20%27Template%20for%20Prod-ESX%20servers%27%22&start=0&count=64",
4.     "category": "server-profile-templates",
5.     "eTag": "2019-03-07T16:09:10.623Z",
6.     "created": "2019-03-07T16:09:10.623Z",
7.     "modified": "2019-03-07T16:09:10.623Z",
8.     "start": 0,
9.     "count": 1,
10.    "total": 1,
11.    "prevPageUri": null,
12.    "nextPageUri": null,
13.    "members": [
14.        {
15.            "type": "ServerProfileTemplateV5",
16.            "uri": "/rest/server-profile-templates/6b279d7a-254c-4cb5-845f-bb50a57c982f",
17.            "name": "Template for Prod-ESX servers",
18.            "description": "",
19.            "serverProfileDescription": "",
20.            "serverHardwareTypeUri": "/rest/server-hardware-types/E9C7E86A-764E-47F2-AA35-EF97134A2600",
21.            "enclosureGroupUri": "/rest/enclosure-groups/d8b4553b-dfb0-4ec7-b58e-97aeea8e776b",
22.            "affinity": "Bay",
23.            "hideUnusedFlexNics": true,
24.            "macType": "Virtual",
25.            "wwnType": "Virtual",
26.            "serialNumberType": "Virtual",
27.            "resetInitialization": "Automatic"
28.        }
29.    ]
30. }
```

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	<a href="https://composer.lj.lab/rest/index/associations">https://composer.lj.lab/rest/index/associations</a>
Parameter key	<code>parentUri</code>
Parameter value	<code>/rest/server-profile-templates/6b279d7a-254c-4cb5-845f-bb50a57c982f</code>
Parameter key	<code>name</code>
Parameter value	<code>server_profile_template_to_server_profiles</code>
Verb	<code>GET</code>
Header	<code>Auth : your sessionID</code> <code>X-API-Version:800</code>



The screenshot shows the Postman interface with a GET request 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`. The Params tab is selected, showing two parameters: `parentUri` and `name`, both with their values highlighted in yellow.

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 screenshot shows a browser-based REST API client interface. At the top, there are tabs for 'Body', 'Cookies', 'Headers (7)', and 'Test Results'. On the right, there are buttons for 'Status: 200 OK', 'Time: 26 ms', 'Size: 778 B', 'Save', and 'Download'. Below these are buttons for 'Pretty', 'Raw', 'Preview', and 'JSON' (with a dropdown arrow). There is also a search icon and a refresh icon.

The main area displays a JSON response with line numbers from 1 to 20. The JSON structure is as follows:

```
1  {
2    "type": "AssociationsV200",
3    "uri": "/rest/index/associations?name=server_profile_template_to_server_profiles&start=0&count=50&parentUri=/rest/server-profile-templates/6b279d7a-254c-4cb5-845f-bb50a57c982f",
4    "category": null,
5    "eTag": null,
6    "created": null,
7    "modified": null,
8    "start": 0,
9    "count": 1,
10   "total": 1,
11   "prevPageUri": null,
12   "nextPageUri": null,
13   "members": [
14     {
15       "parentUri": "/rest/server-profile-templates/6b279d7a-254c-4cb5-845f-bb50a57c982f",
16       "childUri": "/rest/server-profiles/55f81f01-56de-43c8-ba7f-8e4ebcfec780",
17       "name": "server_profile_template_to_server_profiles"
18     }
19   ]
20 }
```

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:

The screenshot shows a browser-based REST API client interface. The JSON response is identical to the one above, but the `childUri` value is highlighted in yellow. A mouse cursor is hovering over the `childUri` value in the JSON response.

```
"members": [
  {
    "parentUri": "/rest/server-profile-templates/6b279d7a-254c-4cb5-845f-bb50a57c982f",
    "childUri": "/rest/server-profiles/55f81f01-56de-43c8-ba7f-8e4ebcfec780",
    "name": "server_profile_template_to_server_profiles"
  }
]
```

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 programming.

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

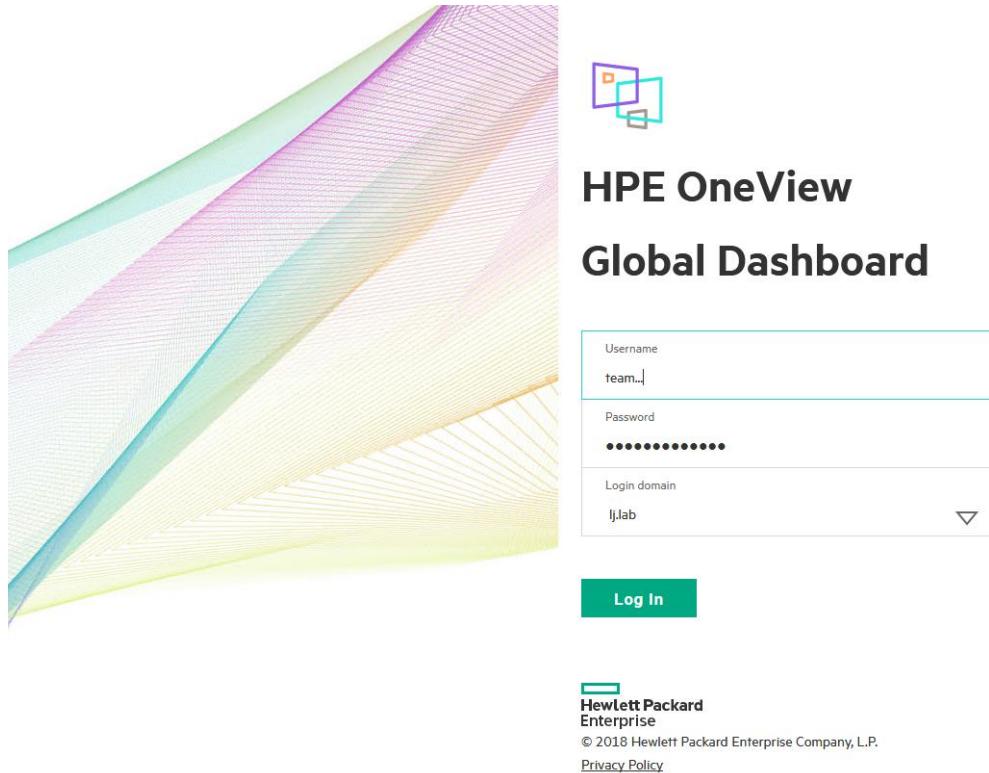
HPE OneView Global Dashboard is a standalone appliance which provides a simple, efficient, and unified view of the health of servers, profiles, and enclosures across multiple HPE OneView appliances, HPE Synergy, HPE 3PAR and SimpliVity systems.

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



The image shows the login interface for the HPE OneView Global Dashboard. On the left is a large, abstract graphic of overlapping colored lines in shades of green, yellow, and purple. To the right of the graphic is the dashboard's logo, featuring three overlapping squares in purple, blue, and orange. Below the logo, the text "HPE OneView" and "Global Dashboard" is displayed in bold, dark font. The login form consists of three input fields: "Username" (team..), "Password" (redacted), and "Login domain" (lj.lab). A "Log In" button is located at the bottom of the form. At the very bottom of the page, there is a footer with the HPE logo, the text "Hewlett Packard Enterprise", "© 2018 Hewlett Packard Enterprise Company, L.P.", and a link to "Privacy Policy".



# Hewlett Packard Enterprise

TSS Paris 2019

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

HPE OneView Global Dashboard

Search

Dashboard

Enclosures Server Hardware Server Profiles Server Profile Templates

Category	OK	Warning	Critical
Enclosures	3	0	0
Server Hardware	24	2	1
Server Profiles	7	0	3
Server Profile Templates	11	0	1

Appliances Appliance Alerts Storage Systems Storage Pools

Category	OK	Warning	Critical
Appliances	2	0	0
Appliance Alerts	12	0	12
Storage Systems	1	0	1
Storage Pools	4	0	0

Volumes

Category	OK	Warning
Volumes	5	2

Select Help

Help

Logout

administrator Infrastructure administrator



Then click on **REST API Reference**

X

Help

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.

- [APPLIANCES](#)
- [CERTIFICATES](#)
- [CONVERGED SYSTEMS](#)
- [ENCLOSURES](#)
- [GROUPS](#)
- [LOGIN SESSIONS](#)
- [MANAGED SANS](#)
- [NETWORK INTERFACES](#)
- [RESOURCE ALERTS](#)
- [SAN MANAGERS](#)
- [SERVER FIRMWARE](#)
- [SERVER HARDWARE](#)
- [SERVER PROFILES](#)
- [SERVER PROFILE TEMPLATES](#)
- [STORAGE POOLS](#)
- [STORAGE SYSTEMS](#)
- [STORAGE VOLUMES](#)

### HPE OneView Global Dashboard REST API (v2)

Download OpenAPI specification: [Download](#)

Product version 1.60.02

#### Appliances

Operations on appliances

#### Add appliance

Add a new appliance into Global Dashboard.

The table below gives an overview of the process for adding an appliance.

Description	Method	Path
Get certificates from remote appliance	GET	/rest/certificates/https/remote/{address}
Store certificates	POST	/rest/certificates/servers
Add appliance	POST	/rest/appliances

POST /rest/appliances

REQUEST SAMPLES

```
{
  "category": "appliances",
  "loginDomain": "local",
  "address": "192.168.0.1",
  "username": "sarah",
  "password": "secret",
  "applianceName": "appliance1"
}
```



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

Click on **Login Sessions**

Search

- APPLIANCES >
- CERTIFICATES >
- CONVERGED SYSTEMS >
- ENCLOSURES >
- GROUPS >
- LOGIN SESSIONS** >
- MANAGED SANS >
- NETWORK INTERFACES >
- RESOURCE ALERTS >
- SAN MANAGERS** >
- SERVER FIRMWARE >
- SERVER HARDWARE >
- SERVER PROFILES >
- SERVER PROFILE TEMPLATES >

**HPE OneView Global Dashboard REST API (v2)**

Download OpenAPI specification: [Download](#)

Product version 1.60.02

## Appliances

Operations on appliances

### Add appliance

Add a new appliance into Global Dashboard.

The table below gives an overview of the process for adding an appliance.

Description	Method	Path
Get certificates from remote appliance	GET	/rest/certificates/https/remote/{address}
Store certificates	POST	/rest/certificates/servers
Add appliance	POST	/rest/appliances



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.

**APPLIANCES**

**CERTIFICATES**

**CONVERGED SYSTEMS**

**ENCLOSURES**

**GROUPS** (1)

**LOGIN SESSIONS**

- Create a session** (2)
- Delete a session**

**PARAMETERS**

**Header Parameters**

- x-api-version: integer [2..2] Required
- content-type: string Required

**REQUEST BODY**

userName: string Required

User name.

**REQUEST SAMPLES**

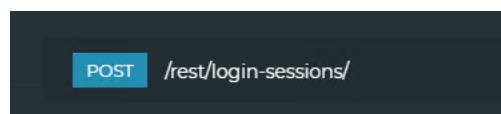
```
{
  "userName: "sarah",
  "password": "secret",
  "authLoginDomain": "local"
}
```

**RESPONSE SAMPLES**

- 200 OK
- 400 Bad request
- 404 Not found
- 412 Precondition failed

```
{
  "token": "NTAxMjQzNDc2NzIyR2yM",
  "user": {
    "domain": "local",
    "user_roles": [ ],
    "userName: "administrator"
  }
}
```

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

### PARAMETERS

#### Header Parameters

x-api-version	integer <b>[ 2 .. 2 ]</b> <b>Required</b> 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.
content-type	string <b>Required</b> Specifies the content type of the request body and should be set to application/json.

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 **UserNme**. As a rule, the API expects lower case words, but capitalized, after the first word, if there are more than one word.

### REQUEST BODY

User credential.

userName	string <b>Required</b> User name.
password	string <b>Required</b> Password of the user.
authLoginDomain	string Login domain of the user. If the user is in LDAP or AD server, then the domain name is the directory server name. If the user is a local user, then the domain name is local.



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

```
REQUEST SAMPLES

{
  "userName": "sarah",
  "password": "secret",
  "authLoginDomain": "local"
}
```

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

Field	Value
URL	<code>https://oneview-global-dashboard.lj.lab/rest/login-sessions</code>
Verb	<code>POST</code>
Header	<code>Content-type:application/json</code> <code>X-API-Version:2</code>
Payload	<code>{     "userName": "xxxxx",     "password": "xxxx",     "authLoginDomain": "lj.lab" }</code>

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



Content of the header:

The screenshot shows the Postman interface for creating a session. The URL is https://oneview-global-dashboard.lj.lab/rest/login-sessions. The 'Headers' tab is selected, showing two entries: Content-Type (application/json) and X-API-Version (2). Other tabs like Params, Authorization, Body, and Tests are also visible.

Content of the body payload:

The screenshot shows the Postman interface for creating a session. The URL is https://oneview-global-dashboard.lj.lab/rest/login-sessions. The 'Body' tab is selected, showing a JSON payload: {"userName": "xxxxx", "password": "xxxxxxxxxxxx", "authLoginDomain": "lj.lab"}. Other tabs like Params, Authorization, Headers, and Tests are also visible.

Press **Send**.



The response we get is the following:

```

1 [
2   "user": {
3     "userName": "team1",
4     "user_roles": [
5       "Infrastructure administrator"
6     ],
7     "domain": "lj.lab"
8   },
9   "token": "ODc0MzA1Mzc0MD0U7UbFjwe4rpe0Bz7a8-z_Qe2w46aI04V",
10  "sessionID": "ODc0MzA1Mzc0MD0U7UbFjwe4rpe0Bz7a8-z_Qe2w46aI04V"
11 }

```

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:

**APPLIANCES** 1

`POST` Add appliance

`GET` Returns a list of appliances 2

`GET` Returns an appliance

`PATCH` Updates an appliance

`DELETE` Remove an appliance

`GET` Returns the appliance single sign-on URL

**CERTIFICATES**

**CONVERGED SYSTEMS**

**ENCLOSURES**

**GROUPS**

**LOGIN SESSIONS**

**MANAGED SANS**

Returns a list of appliances

Returns a list of appliances matching the specified filter. A maximum of 100 resources are returned to the caller per call. Additional calls can be made to this API to retrieve any other appliances matching the filter.

**PARAMETERS**

**Query Parameters**

- `start`: integer  $\geq 0$  Default: 0  
The first item to return, using 0-based indexing. If not specified, the default is 0 - start with the first available item.
- `count`: integer  $\geq 0$   
The number of resources to return. A count of -1 requests all the items up to the maximum page size of 500. 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. If this field is omitted, it defaults to 25.
- `sort`: string  
The sort order of the returned data set. If no sort is

`GET /rest/appliances`

**RESPONSE SAMPLES**

- 200 OK
- 400 Bad request
- 401 Unauthorized
- 412 Precondition failed

```
{
  "category": "appliances",
  "count": 1,
  "created": "2017-10-13T07:26:51Z",
  "members": [
    {
      ...
    }
  ],
  "modified": "2017-10-13T07:26:51Z",
  "nextPageUri": null,
  "prevPageUri": null,
  "start": 0,
  "total": 1,
  "uri": "/rest/appliances?count=1"
}
```



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

### Header Parameters (?)

auth	string <b>Required</b> 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.
x-api-version	integer <b>[2..2] Required</b> 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.

So, use Postman with the parameters found here:

Field	Value
URL	<a href="https://oneview-global-dashboard.lj.lab/rest/appliances">https://oneview-global-dashboard.lj.lab/rest/appliances</a>
Verb	GET
Header	Auth : your sessionID X-API-Version:2



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

Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/> x-api-version	2				
<input checked="" type="checkbox"/> auth	NDA0MDA5MTU0NzM1zc-2yNUESTzv92...				
Key	Value	Description			

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:

```

1 [
2   {
3     "category": "appliances",
4     "count": 2,
5     "members": [
6       {
7         "category": "appliances",
8         "uri": "/rest/appliances/ca3758a0-d368-4e26-8b18-6c53b50eef5f",
9         "applianceUri": "/rest/appliances/ca3758a0-d368-4e26-8b18-6c53b50eef5f",
10        "id": "ca3758a0-d368-4e26-8b18-6c53b50eef5f",
11        "version": "4.10.04-0370820",
12        "versionNumber": 4000100000770820,
13        "currentApiVersion": 800,
14        "hostname": "oneview.lj.lab",
15        "modified": "2019-02-21T09:28:53.457Z",
16        "created": "2018-11-16T10:41:15.551Z",
17        "loginDomain": "Local",
18        "applianceLocation": "192.168.1.10",
19        "applianceName": "HPE OneView",
20        "model": "HPE OneView VM - VMware vSphere",
21        "name": "HPE OneView",
22        "moduleName": "OneView Module",
23        "username": "Administrator",
24        "description": "",
25        "serialNumber": "VMware-564d333079e4f202-0406ae5600481169",
26        "state": "Online",
27        "status": "OK",
28        "groups": [],
29        "eTag": 3,
30        "type": "applianceV1"
31      },
32      {
33        "category": "appliances",
34        "uri": "/rest/appliances/85f94692-facc-460f-86fd-2d3fb91d9f30",
35        "applianceUri": "/rest/appliances/85f94692-facc-460f-86fd-2d3fb91d9f30",
36        "id": "85f94692-facc-460f-86fd-2d3fb91d9f30",
37        "version": "4.10.04-0370820",
38        "versionNumber": 4000100000770820,
39        "currentApiVersion": 800,
40        "hostname": "Composer.lj.lab",
41        "modified": "2019-02-21T09:29:09.489Z",
42        "created": "2018-11-16T10:19:04.005Z",
43        "loginDomain": "Local",
44        "applianceLocation": "192.168.1.110",
45        "applianceName": "Synergy Composer",
46        "model": "Synergy Composer"
47      }
48    ]
49  ]

```



**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	<code>https://oneview-global-dashboard.1j.lab/rest/resource-alerts</code>
Verb	<code>GET</code>
Header	<code>Auth : your sessionID</code> <code>X-API-Version:2</code>

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

```

1 [
2   "category": "alerts",
3   "count": 14,
4   "members": [
5     {
6       "severity": "Critical",
7       "healthCategory": "None",
8       "correctiveAction": "To decode the error-code look into cpqSm2CntrSelfTestErrors description. You can also check iLO log & Diagnostic page to get more details on the error.",
9       "urgency": "None",
10      "changeLog": [],
11      "modified": "2019-02-20T17:03:01.185Z",
12      "associatedEventUris": [
13        "/rest/events/1462434"
14      ],
15      "serviceEventDetails": null,
16      "clearedByUser": null,
17      "alertState": "Active",
18      "assignedToUser": null,
19      "associatedResource": {
20        "resourceCategory": "server-hardware",
21        "resourceName": "Frame1-CH7510600, bay 7",
22        "resourceUri": "/rest/server-hardware/36343537-3338-4E43-3736-303130423659",
23        "associationType": "HAS_A"
24      },
25      "resourceUri": "/rest/server-hardware/36343537-3338-4E43-3736-303130423659",
26      "resourceId": null,
27      "url": "/rest/resource-alerts/c0250109-8156-4e00-a117-8c77a1f3a364",
28      "alertTypeId": "Trap_cpqSm2SelfTestError",
29      "eventSource": "server-hardware",
30      "eTag": "AlertResourceC3",
31      "serviceEventSource": false,
32      "lifecycle": false,
33      "clearedTime": null,
34      "created": "2019-02-20T17:03:01.185Z",
35      "category": "alerts",
36      "eTag": "2019-02-20T17:03:01.185Z",
37      "activityUri": null,
38      "description": "Remote Insight/ Integrated Lights-Out self test error 8192.",
39      "id": "c0250109-8156-4e00-a117-8c77a1f3a364",
40      "originalUri": "/rest/alerts/43323",
41      "state": "Unknown",
42      "applianceUri": "/rest/appliances/85f94692-facc-460f-86fd-2d3fb91d9f30",
43      "applianceName": "Synergy Composer",
44      "applianceLocation": "192.168.1.110",
45      "associatedResourceName": "Frame1-CH7510600, bay 7",
46      "associatedResourceUri": "/rest/server-hardware/36343537-3338-4E43-3736-303130423659"
    }
  ]
]

```



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.

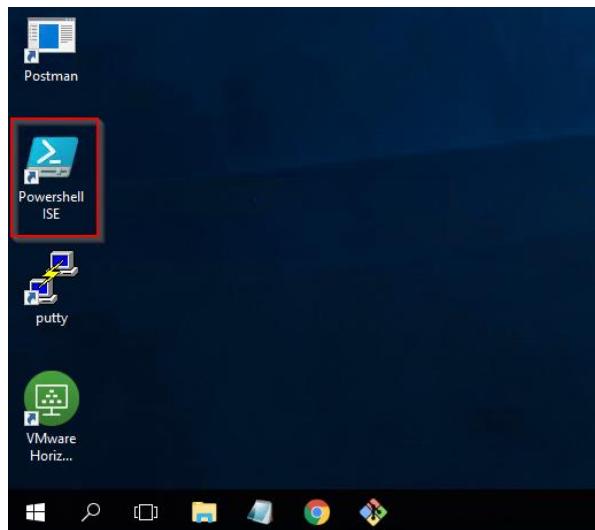
In this lab, we will only glance at two languages, PowerShell and Python. Feel free to jump to the scripting language of your choice.

#### 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.

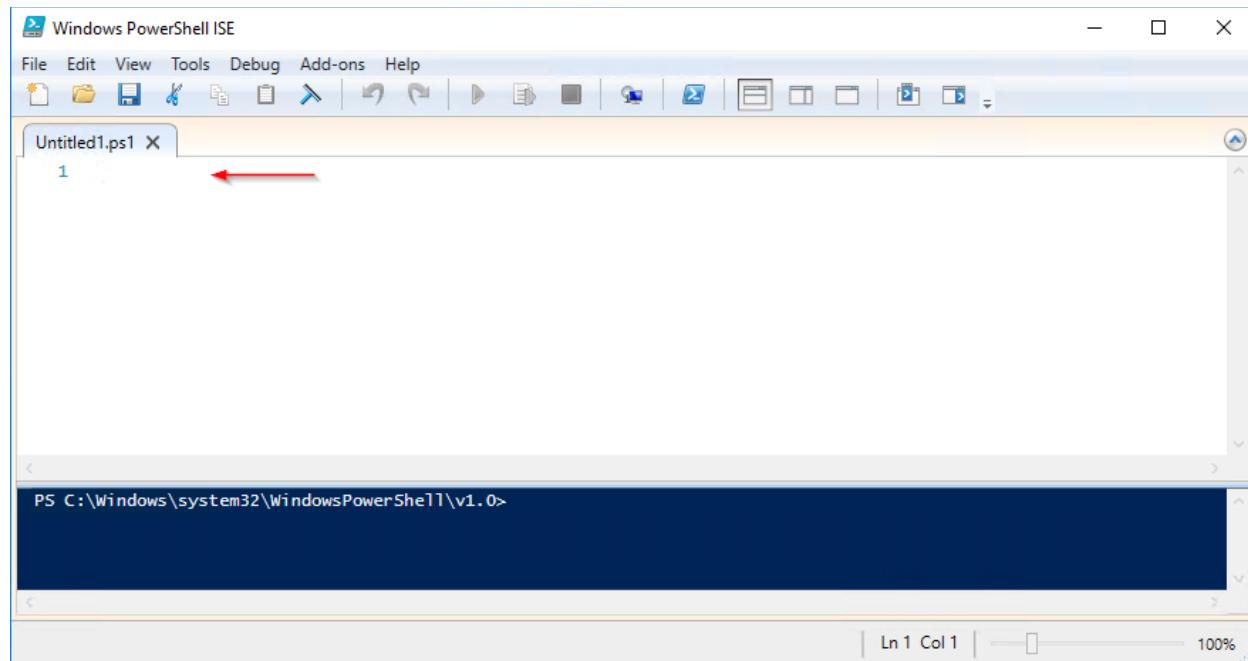
For more information about the HPE OneView PowerShell library, see <https://hewlettpackard.github.io/POSH-HPOneView/>

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 <CTRL> + <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"

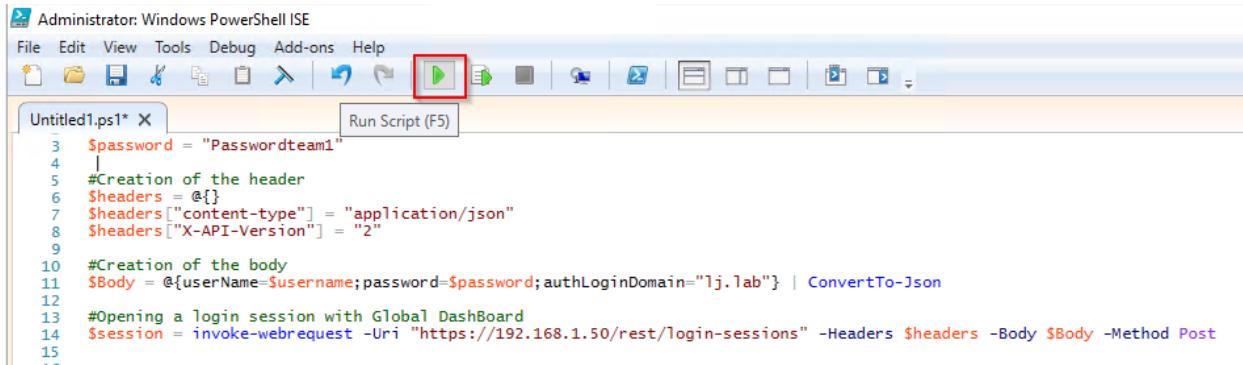
#Creation of the body
$Body = @{userName=$username;password=$password;authLoginDomain="lj.lab"} | ConvertTo-Json

#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.

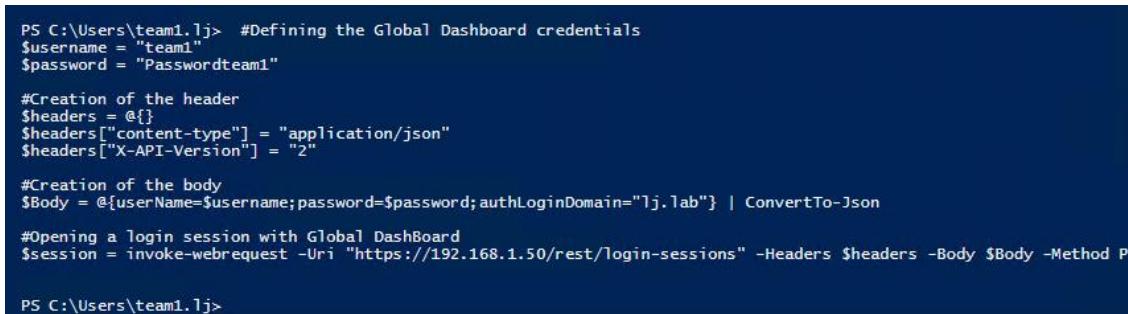


Once properly set, execute the script by pressing on the **Run Script** button



```
Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Untitled1.ps1* X Run Script (F5)
3 $password = "Passwordteam1"
4
5 #Creation of the header
6 $headers = @{}
7 $headers["Content-Type"] = "application/json"
8 $headers["X-API-Version"] = "2"
9
10 #Creation of the body
11 $body = @{username=$username;password=$password;authLoginDomain="lj.lab"} | ConvertTo-Json
12
13 #Opening a login session with Global Dashboard
14 $session = invoke-webrequest -Uri "https://192.168.1.50/rest/login-sessions" -Headers $headers -Body $body -Method Post
15
```

The script should run without error like seen below:



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

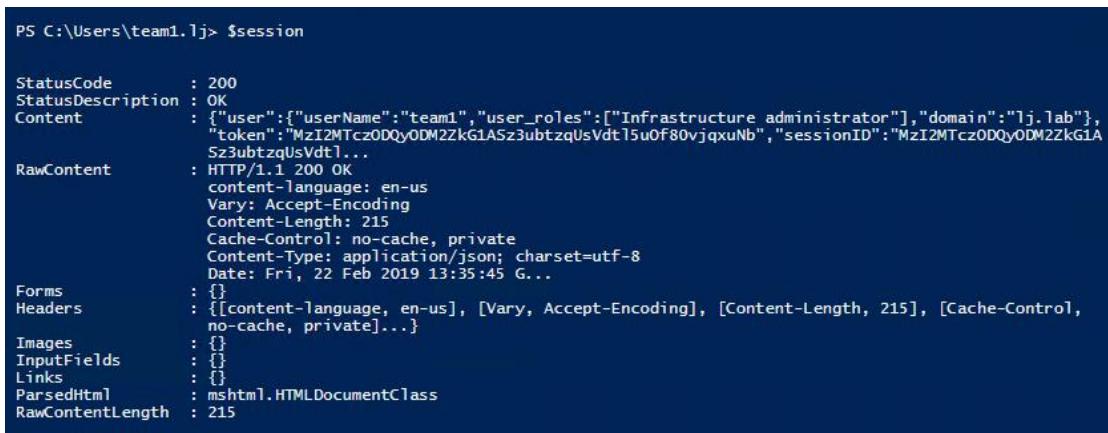
#Creation of the header
$headers = @{}
$headers["Content-Type"] = "application/json"
$headers["X-API-Version"] = "2"

#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 Post

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":"MzI2MTczODQyODM2ZkG1ASz3ubtzqUsVdt15u0f80vjqxuNb","sessionID":"MzI2MTczODQyODM2ZkG1ASz3ubtzqUsVdt1..."}
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\team1.1j>
PS C:\Users\team1.1j> $session.Content
{"user": {"userName": "team1", "user_roles": ["Infrastructure administrator"], "domain": "lj.lab"}, "token": "MzI2MTczODQy
ODM2ZkG1ASz3ubtzqUsVdt15u0f80vjqxuNb", "sessionId": "MzI2MTczODQyODM2ZkG1ASz3ubtzqUsVdt15u0f80vjqxuNb"}
```

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\team1.1j>
PS C:\Users\team1.1j> $session.Content | ConvertFrom-Json
user                               token
-----                               -----
@{userName=team1; user_roles=System.Object[]; domain=lj.lab} MzI2MTczODQyODM2ZkG1ASz3ubtzqUsVdt15u0f80vjqxuNb MzI
ses
sio
nID
---
```

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\team1.1j>
PS C:\Users\team1.1j> ($session.Content | ConvertFrom-Json).token
MzI2MTczODQyODM2ZkG1ASz3ubtzqUsVdt15u0f80vjqxuNb
PS C:\Users\team1.1j>
```

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\team1.1j> $key = ($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\team1.1j> $resourcealerts = (invoke-webrequest -Uri "https://192.168.1.50/rest/resource-alerts" -Headers
PS C:\Users\team1.1j> |
```

The result is saved in `$resourcealerts`

You can retrieve the content of `$resourcealerts` by entering:

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

```
PS C:\Users\team1.1j> $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\team1.1j> $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-d368-4e26-8b18-6c53b50eeef5f
assignedToUser  NoteProperty object assignedToUser=null
associatedEventUris NoteProperty Object[] associatedEventUris=System.Object[]
associatedResource NoteProperty System.Management.Automation.PSCustomObject associatedResource=@{resourceUri=/rest/s...
associatedResourceName NoteProperty string associatedResourceName=iLO-ESX5-0
associatedResourceOriginalUri NoteProperty string associatedResourceOriginalUri=/rest/server-hardware/31373736-3939-4D32-3431-3...
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 t...
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/31373736-3939-4D32-3431-33303437484B
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.mougin.net","uri":"/rest/hypervisor-managers/1dacb777-ff84-41c2-937f-c733cc1445fd"} is disconnected.
correctiveAction : Refer to the hypervisor manager
               {"name":"vcenter.lj.mougin.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/31373736-3939-4D32-3431-333034374B48"} 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-SP2015100_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
```

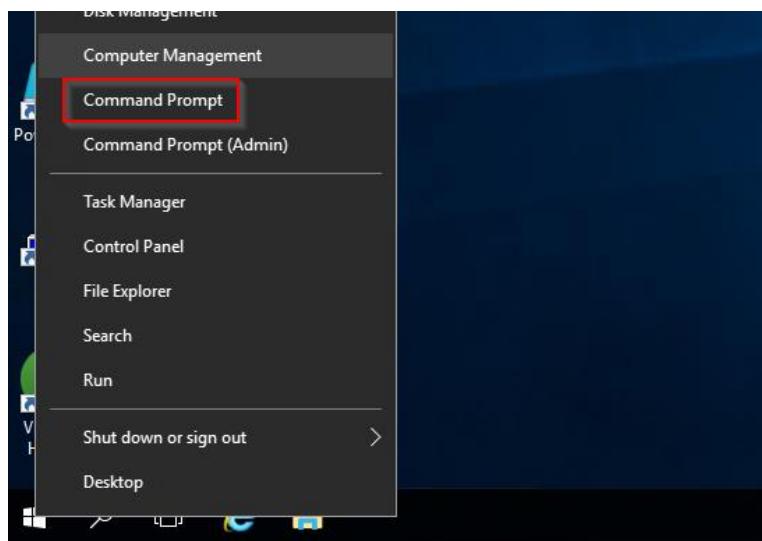


### Python

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**

```
Command Prompt - python
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\team1.lj.000>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> -
```



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
is used. 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": "LTYwOTA20Tcx
Mzgy0h0cAKB_SnAuE9fX1dXnIy4wPubuftCy", "sessionID": "LTYwOTA20TcxMzgy0h0cAKB_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": "LTYwOTA20TcxMzgy0h0cAKB_SnAuE9fX1dXnIy4wPubuftCy",
    "sessionID": "LTYwOTA20TcxMzgy0h0cAKB_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']
'LTYwOTA20TcxMzgy0h0cAKB_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 Self-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 the 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  
the 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 creden  
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>