

HPE Composable Infrastructure Hands-On Lab

Programming 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: Getting started with Postman.....	20
Lab 3: Login Sessions.....	28
Lab 4: API Version and backward compatibility.....	37
Lab 5: API Resource model.....	42
Lab 6: HPE Global Dashboard API	66
Lab 7: Scripting languages	79
PowerShell.....	80
Python.....	100
Summary	120



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

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

Resource operations

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



Table 1. REST HTTP Operations

Operation	HTTP Verb	Description
Create	POST URI <Payload = Resource data>	New resources are created using the POST operation and including relevant data in the payload. On Success the Resource URI is returned.
Read	GET URI	Returns the requested resource representation(s)
Update	PUT URI <Payload= Updatedata>	Update an existing resource using the updatedata.
Delete	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 URL.
<code>/{resource name}</code>		Name of the appliance resources such as server-profiles

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

Data transfer format

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

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



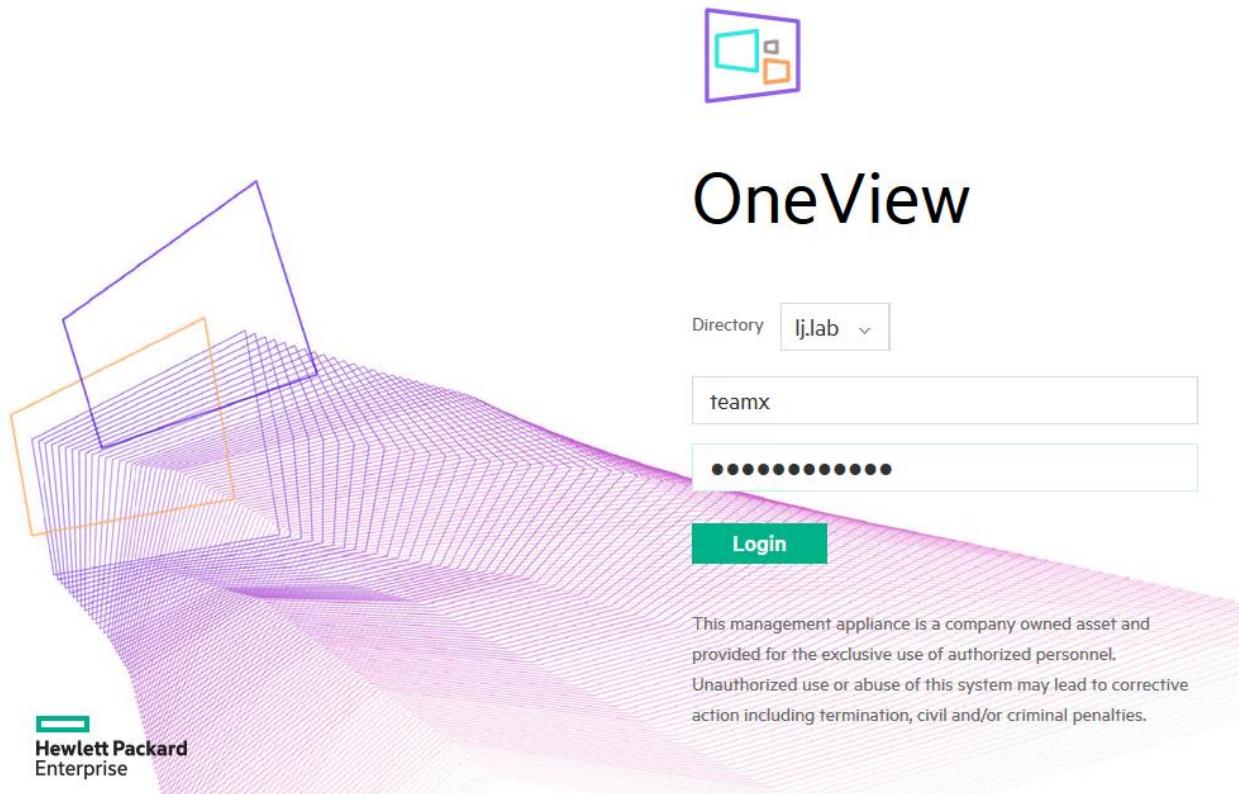
Lab 1: Read the Fantastic Manual

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

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

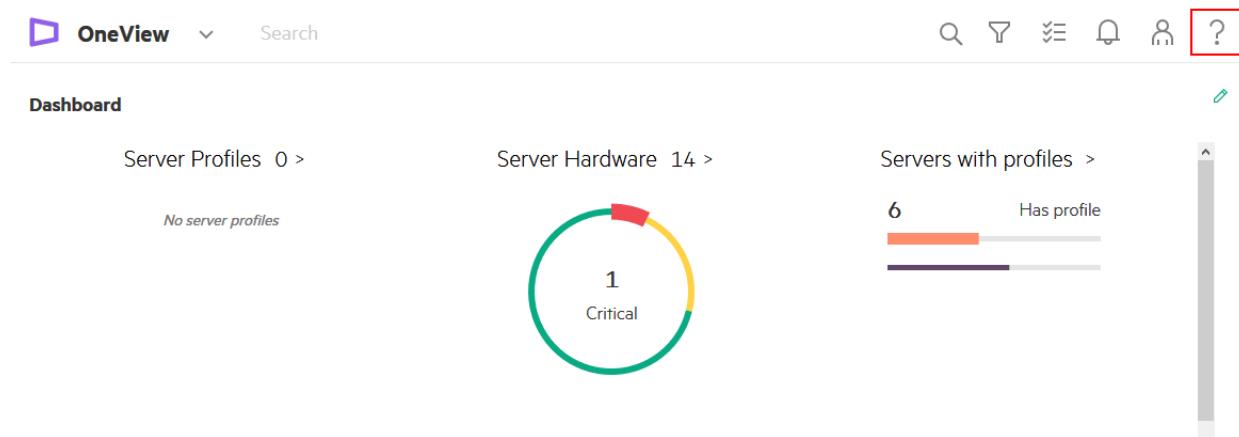
- Directory: **Ij.lab**
- Username: **teamx**
- Password: **Passwordteamx**

with **x** being your team number





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

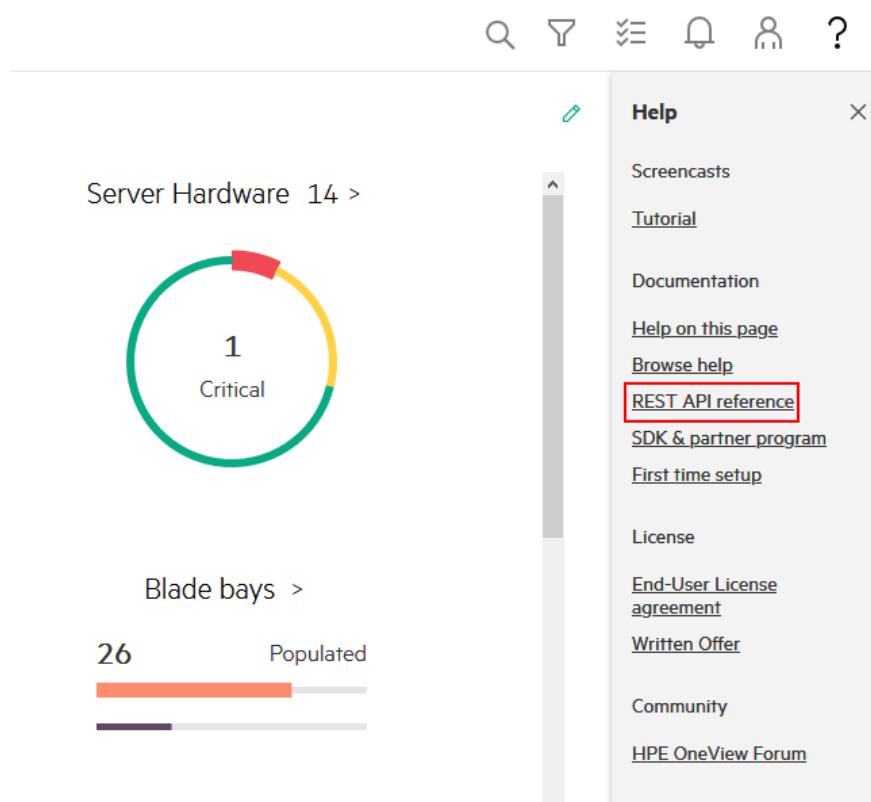


The screenshot shows the HPE OneView dashboard. At the top, there's a navigation bar with icons for OneView, Search, and a help icon (a question mark inside a grey box). Below the navigation is a 'Dashboard' section. It includes three main cards: 'Server Profiles' (0 >), 'Server Hardware' (14 >), and 'Servers with profiles' (6 >). The 'Server Hardware' card features a circular progress bar with a red segment at the top and a yellow segment below it, containing the number '1' and the word 'Critical'. To the right of the cards is a vertical grey scrollbar.

REST APIs are an architectural style following common characteristics and principles; they don't all follow the same standard or specification. Users must really read the documentation to understand how to use the API.

This detailed documentation is found in the *HPE OneView REST API Reference*

Click on the link **REST API Reference**



The screenshot shows the HPE OneView interface with the 'Server Hardware' section selected (14 >). It displays a circular progress bar with a red segment at the top and a yellow segment below it, containing the number '1' and the word 'Critical'. Below this is a 'Blade bays' section with a value of '26' and a 'Populated' status, accompanied by a horizontal progress bar. To the right of the main content is a 'Help' menu sidebar. The 'REST API reference' link in this sidebar is highlighted with a red box.



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

HPE OneView X

API Reference

About

- Security Model
- Response Codes
- Association Names
- Common Parameters
- Common Attributes
- What's New?

► SERVERS

► NETWORKING

► STORAGE

► FC-SANS

► FACILITIES

► EXTERNAL MANAGERS

► HYPERVISORS

► DATA SERVICES

► ACTIVITY

► SETTINGS

► SECURITY

► SEARCH

► REMOTE SUPPORT

► OS-DEPLOYMENT

Search

About the HPE OneView API Reference

The API reference material is used in tandem with the [HPE OneView API scripting documentation](#).

How to use this reference

The table of contents links to the REST API specification for each resource type in the HPE OneView resource model. Each REST API specification defines the REST APIs provided by that resource, including the URI, method, parameters, request body, and response body. In many cases, these components are displayed as links.

These API specifications are intended to provide all the details needed to call the HPE OneView REST APIs and to build scripts around these calls.

Supported REST API versions

This release of HPE OneView supports the REST API version(s) listed in the table below.

A copy of the *API Reference* is available online at www.hpe.com/info/oneview/docs. See also the *REST API Scripting Help* for the current version that is included with the online help for this release.

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



This HPE OneView REST API Reference is also available online, a nice to have when you don't have access to a OneView console

HPE OneView API Reference X

About

- Security Model
- Response Codes
- Association Names
- Common Parameters
- Common Attributes
- What's New?

► SERVERS

► NETWORKING

► STORAGE

► FC-SANS

► FACILITIES

► EXTERNAL MANAGERS

► HYPERVISORS

► DATA SERVICES

► ACTIVITY

► SETTINGS

► SECURITY

► SEARCH

► REMOTE SUPPORT

► OS-DEPLOYMENT

About the HPE OneView API Reference

The API reference material is used in tandem with the [HPE OneView API scripting documentation](#).

How to use this reference

The table of contents links to the REST API specification for each resource type in the HPE OneView resource model. Each REST API specification defines the REST APIs provided by that resource, including the URI, method, parameters, request body, and response body. In many cases, these components are displayed as links.

These API specifications are intended to provide all the details needed to call the HPE OneView REST APIs and to build scripts around these calls.

Supported REST API versions

This release of HPE OneView supports the REST API version(s) listed in the table below.

A copy of the *API Reference* is available online at www.hpe.com/info/oneview/docs. See also the *REST API Scripting Help* for the current version that is included with the online help for this release.

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



Two online copies of the *API Reference* (one is specific to HPE Synergy) are found in the User Guides section:

User Guides (20)	Type	Size	Date
Previous Versions of HPE OneView Documents on HPE Support Center	HTML		
HPE OneView 5.0 User Guide for HPE Synergy	PDF	2.0 MB	Aug 2019
HPE OneView 5.0 User Guide	PDF	1.3 MB	Aug 2019
HPE OneView 5.0 Help for HPE Synergy	HTML	N/A	Aug 2019
HPE OneView 5.0 Help	HTML	N/A	Aug 2019
HPE OneView 5.0 API Reference for HPE Synergy (API version 1200)	HTML	N/A	Aug 2019
HPE OneView 5.0 API Reference (API version 1200)	HTML	N/A	Aug 2019
HPE OneView 4.2 User Guide for HPE Synergy	PDF	2.0 MB	Feb 2019

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

About the HPE OneView API Reference

The API reference material is used in tandem with the [HPE OneView API scripting documentation](#).

How to use this reference

The table of contents links to the REST API specification for each resource type in the HPE OneView resource model. Each REST API specification defines the REST APIs provided by that resource, including the URI, method, parameters, request body, and response body. In many cases, these components are displayed as links.

These API specifications are intended to provide all the details needed to call the HPE OneView REST APIs and to build scripts around these calls.

Supported REST API versions

This release of HPE OneView supports the REST API version(s) listed in the table below.

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



The group above provides important information about the API (supported versions), security model (authentication and authorization), association 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.

The screenshot shows the 'About' section of the HPE OneView API Reference. A red box highlights the following menu items: Security Model, Response Codes, Association Names, Common Parameters, Common Attributes, and What's New?. Below this, there is a sidebar with navigation links for SERVERS, NETWORKING, STORAGE, and FC-SANS.

- ▶ SERVERS
- ▶ NETWORKING
- ▶ STORAGE
- ▶ FC-SANS

Search

About the HPE OneView API Reference

The API reference material is used in tandem with the [HPE OneView API scripting documentation](#).

How to use this reference

The table of contents links to the REST API specification for each resource type in the HPE OneView resource model. Each REST API specification defines the REST APIs provided by that resource, including the URI, method, parameters, request body, and response body. In many cases, these components are displayed as links.

These API specifications are intended to provide all the details needed to call the HPE OneView REST APIs and to build scripts around these calls.

Supported REST API versions

Navigate rapidly through the different menus to explore their content.



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

The screenshot shows the HPE OneView API Reference interface. On the left, there's a sidebar with a dark blue header containing the title "HPE OneView API Reference". Below the header, the sidebar has a "About" section with links to "Security Model", "Response Codes", "Association Names", "Common Parameters", "Common Attributes", and "What's New?". Under a "▼ SERVERS" section, there are links to "Connections", "Enclosure Groups", and "Enclosures". The "Enclosures" link is highlighted with a red border. At the bottom of the sidebar, there's a link to "ID Pools". The main content area has a white background with a search bar at the top right labeled "Search". The main content starts with a section titled "About the HPE OneView API Reference", which includes a note about using it with the [HPE OneView API scripting documentation](#). Below this is a "How to use this reference" section that explains the REST API specifications for resources. The next section, "Supported REST API versions", notes that the release supports multiple versions. The entire screenshot is framed by a thin gray border.

About the HPE OneView API Reference

The API reference material is used in tandem with the [HPE OneView API scripting documentation](#).

How to use this reference

The table of contents links to the REST API specification for each resource type in the HPE OneView resource model. Each REST API specification defines the REST APIs provided by that resource, including the URI, method, parameters, request body, and response body. In many cases, these components are displayed as links.

These API specifications are intended to provide all the details needed to call the HPE OneView REST APIs and to build scripts around these calls.

Supported REST API versions

This release of HPE OneView supports the REST API version(s) listed in the table below.



The following is displayed:

The screenshot shows the HPE OneView API Reference interface. On the left, there's a sidebar with a dark blue header containing 'HPE OneView' and 'API Reference'. Below this are links for 'About', 'Security Model', 'Response Codes', 'Association Names', 'Common Parameters', 'Common Attributes', and 'What's New?'. Under 'SERVERS', there are links for 'Connections', 'Enclosure Groups', and 'Enclosures' (which is highlighted in a darker shade). Below 'Enclosures' are links for 'ID Pools', 'ID Pools IPv4 Ranges', 'ID Pools IPv4 Subnets', and 'ID Pools IPv6 Ranges'. The main content area has a light gray header with 'Search' and a magnifying glass icon. The title 'Enclosures' is bolded. A descriptive paragraph follows, mentioning that the enclosures resource provides REST APIs for managing enclosures. It details how to retrieve, add, and remove enclosures, along with filter parameters and descriptions for each API method: GET /rest/enclosures, POST /rest/enclosures, GET /rest/enclosures/{id}, PATCH /rest/enclosures/{id}, and DELETE /rest/enclosures/{id}.

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: 1200
```

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: 1200
```

 **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: 1200

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

```

Headers → Content-Type, Auth, X-Api-Version

Body → { "hostname" : "fe80::1:2:3:4", }

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

Request Headers

Accept-Language	The language code requested in the response. If a suitable match to the requested language is not available, en-US or the appliance locale is used.	string
Auth	Session authorization token obtained from logging in . If this header is not included or if the session-token is invalid, the response code will be 401 Unauthorized.	string required
Content-Type	The data format sent in the request body. If the Content-Type header is not provided, application/octet-stream is assumed. Any value other than the required value will result in a response code of 415 Unsupported Media Type.	string required
	Required Value	application/json
X-Api-Version	Specifies the version of the API to invoke. The behavior of a given API version remains the same. It is upward compatible from release to release. New versions of an API are created when new features or changes are introduced. To ensure expected behavior, always provide the X-Api-Version value.	integer required



- Then depending of the type of invocation we have:

- With GET:

- **Query parameters:** describing the different query options: start, sort, count, fields, filter, query, etc.:

Query Parameters

count	The number of resources to return. A count of -1 requests all the items. The actual number of items in the response can differ from the requested count if the sum of start and count exceed the total number of items, or if returning the requested number of items would take too long.	integer
See the Common Parameters page for more information on the usage of count.		
Default	-1	
fields	Specifies which fields should be returned in the result set.	string
filter	A general filter/query string to narrow the list of items returned. The default is no filter - all resources are returned.	
See the Common Parameters page for more information on the usage of filter.		
array of string		
query	A general query string to narrow the list of resources returned. The default is no query - all resources are returned.	string
See the Common Parameters 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	
category	Identifies the resource type	string read only
count	The actual number of resources returned in the specified page	integer
created	Date and time when the resource was created	timestamp read only
Format	yyyy-MM-dd'T'HH:mm:ss.SSSZ'	
Pattern	[I-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	
eTag	Entity tag/version ID of the resource, the same value that is returned in the ETag header on a GET of the resource	string read only
members	List of enclosures resources. array of EnclosureV7	
applianceBayCount	The number of appliance bays in the enclosure.	integer read only
applianceBays	A list of the appliance bays in the enclosure.	Appliance read only
bayNumber	The bay number of the appliance.	integer read only
bayPowerState	The power state of the appliance bay.	



- With POST/PUT/PATCH/DELETE:
 - **Response header:** providing the newly created resource or task created to perform the requested action.
- With POST/PUT/PATCH/DELETE:
 - **Request Body:** describing the content of the body that needs to be provided)

Request Body	array of PATCH Operation	
from	The JSON path to use as the source of a "move" or "copy" operation. Not used by "add", "remove", "replace", or "test".	string read only
op	The type of operation: one of "add", "copy", "move", "remove", "replace", or "test".	string required read only
path	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 links: About, Security Model, Response Codes, Association Names, Common Parameters, and Common Attributes. The main area has a search bar at the top with the word "version". Below the search bar, a list of API endpoints is displayed, each with a small description. The first endpoint, "GET /rest/version", is highlighted with a red rectangular box around its link text.

API Endpoint	Description
<code>GET /rest/version</code>	
<code>GET /rest/appliance/nodeinfo/version</code>	
<code>POST /rest/appliance/eula/save</code>	
<code>GET /rest/appliance/firmware/notification</code>	
<code>GET /rest/appliance/firmware/pending</code>	
<code>POST /rest/appliance/firmware/image</code>	



You will find something like this:

The screenshot shows the HPE OneView API Reference interface. On the left, there's a sidebar with a dark blue header "HPE OneView API Reference" and a list of endpoints: Backups, Diagnostic Service Sessions, Configuration, Domains, Email notification, Firmware Bundles, Firmware Drivers, Global Settings, HA Nodes, and Hardware Appliance. The main content area has a title "GET /rest/version". Below it, a "GET" method is selected, and the URL "/rest/version" is shown. A detailed description follows: "Returns the range of possible API versions supported by the appliance. The response contains the current version and the minimum version. The current version is the recommended version to specify in the REST header. The other versions are supported for backward compatibility, but might not support the most current features." A "Get Supported API Versions" button is present. Further down, another description states: "Retrieve the minimum and current version of the supported REST API versions. The current version should be used for all new REST API calls and is specified in the request X-Api-Version header." At the bottom of the main content area, there's a "Request" section.

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" : 1200,
  "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 like 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: Getting started with Postman

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 several 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 useful features like:

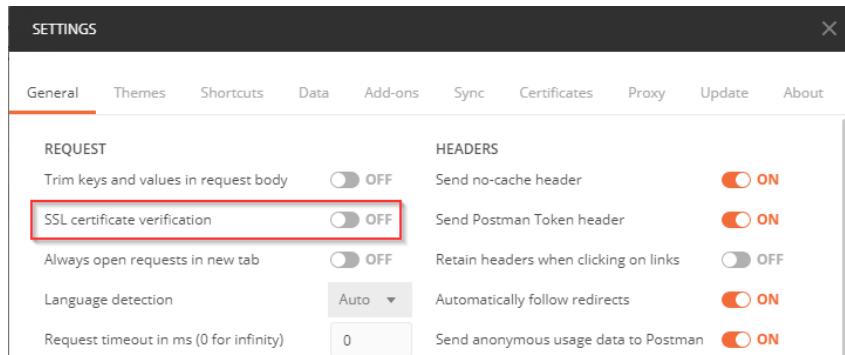
- You can save your REST calls and share the collections with others
- You can use variables to store for instance the OneView authentication session keys

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



Note: Postman can be downloaded from <https://dl.pstmn.io/download/latest/win?arch=64>

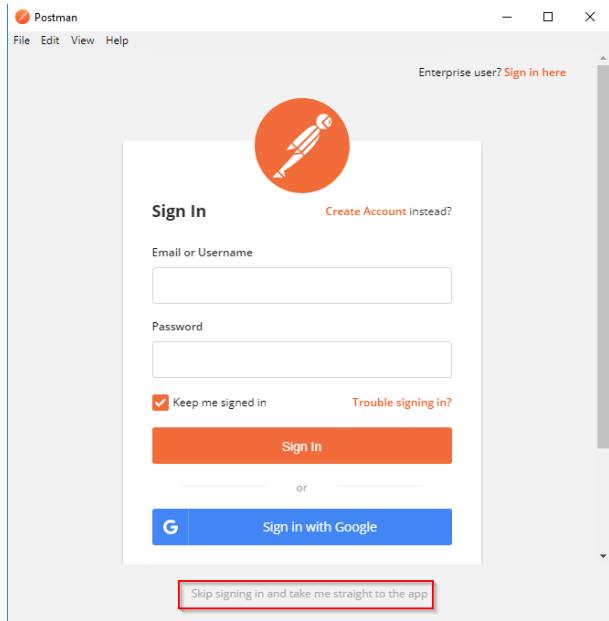
Note: In Settings, you must set **SSL certificate verification** to **OFF** if OneView uses a self-signed certificate



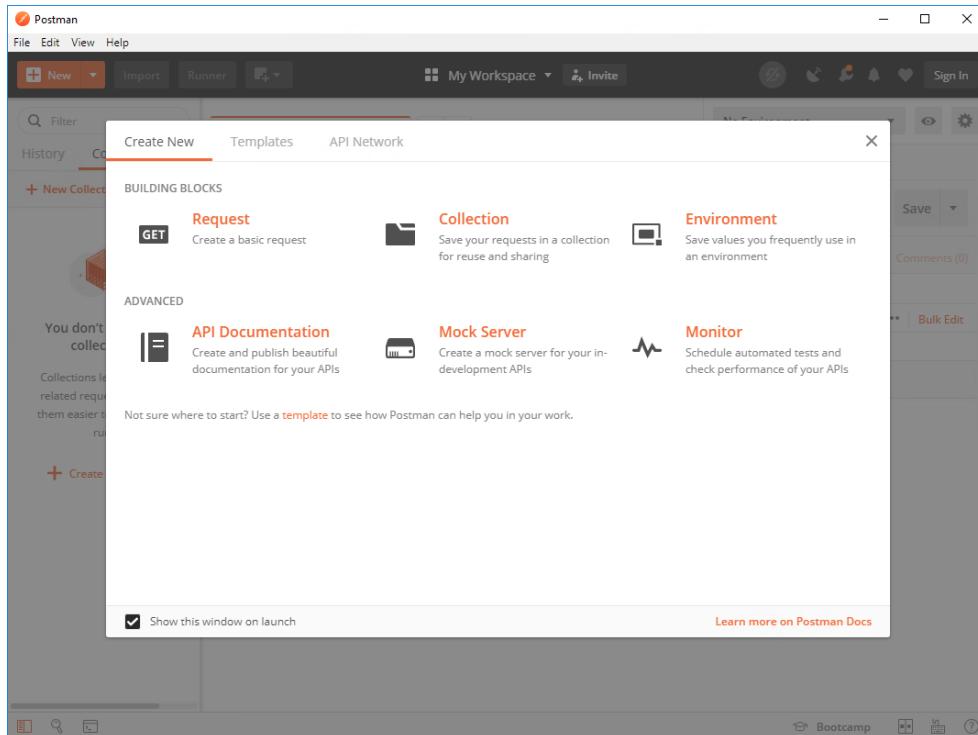
Note: If you are not very comfortable with Postman, two Postman collections are available in e:\Postman\ with all the REST calls we are creating in these labs. If these collections are not already in the Postman Collections pane, click the Import button in the header bar and go to e:\Postman then select the two files to import (*OneView Lab collection.postman_collection.json* and *Global DashBoard collection.postman_collection.json*)



If the following window opens, you must click on **Skip signing in and take me straight to the app** at the bottom to start Postman without signing:



Then you can close the following window:





Before using Postman, let's take a quick look at what the main pane provides. There are two main sections, on the upper one, we find where we can enter the query information (verb, URL, header/body, etc.):

The screenshot shows the Postman interface with the following details:

- Verb:** GET
- URL:** https://composer.lj.mougin.net/rest/enclosures
- Headers:**
 - X-API-Version: 800
 - Auth: {{sessionID}}

And on the lower section is where the result of the REST query resides with the body response, the response status, etc.:

The screenshot shows the Postman interface with the following details:

- Status code:** 200 OK
- Response:** JSON output showing an EnclosureListV7 object with members and their properties.

```

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

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

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

Back to Postman, click the **New** button in the header bar and select **Request** to create our first API query:

The screenshot shows the Postman application window. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'New' (highlighted with a red circle 1), 'Import', 'Runner', and a user icon. The main workspace is titled 'My Workspace' with an 'Invite' button. A sidebar on the left lists 'BUILDING BLOCKS' with 'Request' highlighted (red circle 2) and other options like 'Collection', 'Environment', 'Documentation', 'Mock Server', and 'Monitor'. The central area shows an 'Untitled Request' with a 'GET' method selected. Below it, the 'Params' tab is active, showing a table with one row: 'Key' (Value: 'Key') and 'Value' (Value: 'Value'). Other tabs include 'Authorization', 'Headers', 'Body', 'Pre-request Script', 'Tests', 'Cookies', 'Code', and 'Comments (0)'. The bottom section is labeled 'Response'.



Enter a request name like **OneView Version** then click **Create Collection** and enter a name like **OneView collection**:

SAVE REQUEST X

Learn more about creating collections

Request name 1 OneView version

Request description (Optional)
Adding a description makes your docs better

Descriptions support Markdown

Select a collection or folder to save to:

Search for a collection or folder 2

All Collections 3 OneView collection 4

Cancel Save 5

Click on the validate icon ✓ to save the collection name then select your new collection and click **Save to OneView collection**.



To display the new collection, you can select **Collections** on the left-hand pane:

The screenshot shows the Postman application window. The top navigation bar includes File, Edit, View, Help, New, Import, Runner, My Workspace, Invite, and Sign In. The left sidebar has History, Collections (which is highlighted with a red box), and Trash. A 'OneView collection' is listed under History with 1 request. The main workspace shows a 'OneView version' collection with a GET request to 'OneView version'. The request details panel shows 'Params', 'Authorization', 'Headers', 'Body', 'Pre-request Script', 'Tests', 'Cookies', 'Code', and 'Comments (0)'. Below the request details is a table for parameters, with one row showing 'Key' and 'Value'. The bottom section is labeled 'Response'.

Then you can enter the following information to create your first query:

- 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 'OneView Version' collection in Postman. The request details panel has 'GET' selected (1) and the URL 'https://composer.lj.lab/rest/version' entered (2). The 'Headers' tab is selected (3). Under Headers, there is a table with one row: 'Accept' checked (4) with value 'application/json' (5). The 'Send' button is highlighted with a red arrow (6).

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 a browser interface with a header bar containing 'Body', 'Cookies', 'Headers (12)', 'Test Results', 'Status: 200 OK' (which is highlighted with a red box), 'Time: 42ms', 'Size: 391 B', and 'Save Response'. Below the header is a main content area titled 'REST API Response Codes' with a table listing various status codes and their details.

Unsuccessful status code can be *400 Bad request*, *401 Unauthorized*, *415 Unsupported Media Type*, etc. The list of all status codes with some troubleshooting indications is available in the OneView API Reference. Those indications can be useful for troubleshooting when your request is failing:

The screenshot shows the 'HPE OneView API Reference' page. On the left is a sidebar with links like 'About', 'Security Model', 'Response Codes' (which is highlighted with a red box), 'Association Names', 'Common Parameters', 'Common Attributes', 'What's New?', and sections for '▶ SERVERS', '▶ NETWORKING', '▶ STORAGE', '▶ FC-SANS', '▶ FACILITIES', and '▶ EXTERNAL MANAGERS'. The main content area is titled 'REST API Response Codes' and contains a table with the following data:

Status Code	Scenario	Methods	Response body
200 OK	Successful return from a synchronous read/query operation. The URI points to a valid resource or collection, but there is nothing to return based on specified filters. Successful return from a synchronous update operation. Successful (synchronous) POST returned information that is not identified by a URI. The DELETE method typically returns 204 NO CONTENT, not 200 OK.	GET PUT, PATCH POST	An entity (or list of entities) corresponding to the requested resource(s). Empty list. An entity (or list of entities) corresponding to the updated resource(s). An entity that describes the result of the operation.
201 CREATED	Successful return from a synchronous POST.	POST	An entity (or list of entities) corresponding to the created resource.

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 easily that `currentVersion=1200` (meaning HPE OneView v5.0)

The screenshot shows a browser interface with a header bar containing 'Body', 'Cookies', 'Headers (12)', 'Test Results', 'Status: 200 OK' (highlighted with a red box), 'Time: 42ms', 'Size: 391 B', and 'Save Response'. Below the header is a main content area showing a JSON response body:

```

1  {
2    "minimumVersion": 120,
3    "currentVersion": 1200
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, let's check the syntax of the login session by selecting **Security / Login Sessions** in the API Reference.

This is what we find:

The screenshot shows the HPE OneView API Reference interface. On the left, there is a sidebar with a dark blue background containing a navigation menu. The menu items include: DATA SERVICES, ACTIVITY, SETTINGS, SECURITY (with a red circle containing the number 1), Active User Sessions, Appliance Certificates, Authorizations, Certificate Authority, Certificate Validation, Configuration, Certificates - Client, Certificates - Server, Certificates Client RabbitMq, Login Details, Login Domains, Login Domains Global Settings, Login Domains Group to Role Mapping, Login Sessions (with a red circle containing the number 2), Roles, and Secure Data at Rest. The 'Login Sessions' item is currently selected, highlighted with a red border. The main content area has a white background and contains the following information:

Login Sessions

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.

▶ **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. Specified role name should match with any of the role names returned by GET /rest/roles API and is case sensitive. If scopeUri ...

▶ **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 provide client authentication in addition to the normal server authentication during TLS negotiation. Here the private key will be used by the ...



If you expand the first POST, we 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: 1200
Content-Type: application/json

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

← Header

← Body / Payload

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 **1200**)
- **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 username to use for authentication
- **password**: set to the password for that username



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 provided parameters:

Field	Value
URL	<code>https://{{app1}}/rest/login-sessions</code>
Verb	<code>POST</code>
Header	<code>Content-type:application/json</code> <code>X-API-Version:1200</code>
Payload	<pre>{ "authLoginDomain": "mydirectory", "password": "mypassword", "userName": "administrator", "loginMsgAck": "true" }</pre>

So, click on **+** to create a new request

The screenshot shows the Postman interface with a new request creation dialog. The dialog has a red box around the '+' button. The URL field contains `https://composer.lj.lab/rest/version`. The Headers tab is selected, showing two entries: 'Accept' with value 'application/json' and 'Key' with value 'Value'. Other tabs include Params, Authorization, Body, Pre-request Script, and Tests.



And enter the following information:

- In ① Select in Verb: **Post**
- In ② Enter the URL: **<https://composer.lj.lab/rest/login-sessions>**

Content of the headers:

- In ③ Select **Headers**
- Enter in key ④: **Content-type**
- Enter in value ⑤: **application/json**
- Enter in key ⑥: **x-api-version**
- Enter in value ⑦: **1200**
-

The screenshot shows the Postman interface with a POST request to <https://composer.lj.lab/rest/login-sessions>. The Headers tab is selected, displaying the following configuration:

KEY	VALUE	DESCRIPTION
Content-Type	application/json	
x-api-version	1200	
Key	Value	Description

Content of the Body:

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

```
{  
  "authLoginDomain": "lj.lab",  
  "password": "<mypassword>",  
  "userName": "<myusername>"  
}
```



The screenshot shows the Postman interface with a POST request to `https://composer.lj.lab/rest/login-sessions`. The 'Body' tab is selected, containing the following JSON payload:

```
1 {
2 "authLoginDomain": "lj.lab",
3 "password": "Passwordteam1",
4 "userName": "team1"
5 }
6
```

Then press **Send**.

The response we get is the following:

The screenshot shows the Postman response view with a status of 200 OK. The response body is:

```
1 {
2   "sessionID": "NDIxMDIzODg0NTU4D04s46gggCKcY1s09MhZon5HdIKAB9nd",
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 must 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** in the search pane and select `GET /rest/global-settings`



This is what we can read:

The screenshot shows the HPE OneView API Reference interface. On the left, there's a sidebar with various categories like Appliance SSH Access, Appliance State, etc., and a 'Global Settings' section which is currently selected. The main content area is titled 'Global Settings' and describes it as a resource for storing name/value pairs. It includes a 'GET /rest/global-settings' endpoint description, a 'List Settings' example, and a 'Request' block with a sample GET request and its headers.

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: 1200
```

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

So, before we use Postman with the parameters found here, it is necessary to record your **SessionID** from your **POST /rest/login-sessions**:

The screenshot shows a Postman request response. The JSON body contains a session ID. A context menu is open over the session ID field, with 'Copy' highlighted. Other options in the menu include Undo, Redo, Cut, Paste, and Select All. Below the menu, there are links for Find, EncodeURIComponent, and DecodeURIComponent.

Status: 200 OK Time: 920ms Size: 428 B

Body Cookies Headers (12) Test Results

Pretty Raw Preview Visualize **BETA** JSON ▾

```
1 {  
2   "sessionId": "NDIxMDIzODg0NTU4DO4s46gggCKcY1s09MhZon5HdIKAB9r-1"  
3   "partnerData": {}  
4 }
```

Undo
Redo
Cut
Copy
Paste
Select All

Find: NDIxMDIzODg0NTU4DO4s46gggCKcY1s09MhZon5HdIKAB9r-1
EncodeURIComponent
DecodeURIComponent



The screenshot shows the Postman interface with a GET request to <https://composer.lj.mougin.net/rest/global-settings>. The Headers tab is selected, showing two parameters: **x-api-version** with value 800, and **Auth** with value **LTM5ODkzNjE0MTQyxfU_eCIWFGp_BlaEGaGINM...**. A red arrow points from the text "SessionID" to the Auth header value.

Then to create the new request, you can use the **Duplicate Tab** option to duplicate your *POST /rest/login-sessions* so that you don't have to re-enter the default headers parameters again:

The screenshot shows the Postman interface with a POST request to <https://composer.lj.lab/rest/login-sessions>. The Headers tab is selected, showing 10 parameters. A context menu is open over the Headers tab, with the **Duplicate Tab** option highlighted by a red box.

and change the parameters using the following configuration:

Field	Value
URL	https://composer.lj.lab/rest/global-settings

Verb	GET
Header	Auth : <your sessionID>

The screenshot shows the Postman interface with a GET request to <https://composer.lj.lab/rest/global-settings>. The Headers tab is selected, showing 11 parameters. The **auth** parameter has the value **NDIxMDIzODg0NTU4DO4s46gggCKcY1s09MhZon5HdIKAB9nd**, which is highlighted by a red box.

Then when all set, press **Send**



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

Body Cookies Headers (7) Test Results

Pretty Raw Preview JSON

```
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     }
```



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

The screenshot shows a REST API testing interface. At the top, there's a table for 'Headers (3)' with columns for KEY, VALUE, and DESCRIPTION. Three headers are listed: Content-Type (application/json), x-api-version (1200), and auth (NDlxMDIzODg0NTU4DO4s46gggCKcY1s09MhZon5HdIKAB9nd). Below this is a section for 'Temporary Headers (8)'. At the bottom, tabs for Body, Cookies, Headers (12), and Test Results are visible, along with status information: Status: 401 Unauthorized, Time: 15ms, Size: 648 B, and Save Response.

Headers (3)

KEY	VALUE	DESCRIPTION
Content-Type	application/json	
x-api-version	1200	
auth	NDlxMDIzODg0NTU4DO4s46gggCKcY1s09MhZon5HdIKAB9nd	

Temporary Headers (8) ①

Body Cookies Headers (12) Test Results Status: 401 Unauthorized Time: 15ms Size: 648 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
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 }
```

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**

The screenshot shows the 'About' section of the HPE OneView API Reference. On the left, there is a sidebar with links: 'About' (which is selected and highlighted in red), 'Security Model', 'Response Codes', 'Association Names', 'Common Parameters', 'Common Attributes', and 'What's New?'. The main content area has a search bar at the top right. Below the search bar, the title 'Global Settings' is displayed. A brief description follows: '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.' Underneath this, there is a 'GET /rest/global-settings' entry with a description: 'Gets a list of settings for all the authorized categories based on optional sorting & filtering and constrained by start and count parameters'. At the bottom of this section is a link labeled 'List Settings'.

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:

Supported REST API versions

This release of HPE OneView supports the REST API version(s) listed in the table below.

A copy of the *API Reference* is available online at www.hpe.com/info/oneview/docs. See also the *REST API Scripting Help* for the current version that is included with the online help for this 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
4.20	1000
5.00	1200

You can see that 1200 is the API version for OneView 5.0.

The second table lists the API versions that are no longer supported in our release version (5.0)

Note: The following API versions are *no longer supported* in this release

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

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

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



So, use Postman with the following parameters:

Field	Value
URL	https://composer.lj.lab/rest/ethernet-networks
Verb	GET
Header	Auth : your sessionID X-API-Version:1200

The body response is the following:

GET https://composer.lj.lab/rest/ethernet-networks Send Save

Params Authorization Headers (11) Body Pre-request Script Tests Settings Cookies Code Comments (0)

Headers (3)

KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
Content-Type	application/json				
x-api-version	1200				
auth	NDlxMDIzODg0NTU4DO4s46gggCKcY1s09MhZon5HdIKAB9nd				
Key	Value	Description			

Temporary Headers (8)

Body Cookies Headers (12) Test Results Status: 200 OK Time: 51ms Size: 9.9 KB Save Response

Pretty Raw Preview Visualize BETA JSON

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
      {
        "type": "NetworkCollectionV4",
        "uri": "/rest/ethernet-networks?start=0&count=13",
        "category": "ethernet-networks",
        "eTag": null,
        "created": null,
        "modified": null,
        "start": 0,
        "count": 13,
        "total": 13,
        "prevPageUri": null,
        "nextPageUri": null,
        "members": [
          {
            "type": "ethernet-networkV4",
            "uri": "/rest/ethernet-networks/125e2990-4efa-4800-a4d9-6ce2ed6253ca",
            "category": "ethernet-networks",
            "eTag": "bb28d013-7414-4705-af60-b33d84a3fda0",
            "created": "2019-12-23T14:34:26.729Z",
            "modified": "2019-12-23T14:34:26.739Z",
            "scopesUri": "/rest/scopes/resources/rest/ethernet-networks/125e2990-4efa-4800-a4d9-6ce2ed6253ca",
            "vlanId": 881,
            "subnetUri": null,
            "ipv6SubnetUri": null,
            "-----"
          }
        ]
      }
    
```

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 1200:

GET https://composer.lj.lab/rest/ethernet-networks

Headers (11)

KEY	VALUE	DESCRIPTION
Content-Type	application/json	
x-api-version	300	
auth	NDlxMDIzODg0NTU4DO4s46gggCKcY1s09MhZo...	
Key	Value	Description

Temporary Headers (8)

Body Cookies Headers (12) Test Results

Status: 200 OK Time: 33ms Size: 8.43 KB Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "type": "NetworkCollectionV300",
3   "uri": "/rest/ethernet-networks?start=0&count=13",
4   "category": "ethernet-networks",
5   "eTag": null,
6   "created": null,
7   "modified": null,
8   "start": 0,
9   "count": 13,
10  "total": 13,
11  "prevPageUri": null,
12  "nextPageUri": null,
13  "members": [
14    {
15      "type": "ethernet-networkV300",
16      "uri": "/rest/ethernet-networks/125e2990-4efa-4800-a4d9-6ce2ed6253ca",
17      "category": "ethernet-networks",
18      "eTag": "bb28d013-7414-4705-af60-b33d84a3fda0",
19      "created": "2019-12-23T14:34:26.729Z",
20      "modified": "2019-12-23T14:34:26.739Z",
21      "vlanId": 881,
22      "subnetUri": null,
23      "connectionTemplateUri": "/rest/connection-templates/71a42538-d888-4e2e-a0b6-a1847cc6b9dd",
24      "privateNetwork": false,
25      "smartLink": true,
26      "purpose": "General",
27      "ethernetNetworkType": "Tagged",
28      "fabricUri": "/rest/fabrics/d3e64cd6-f176-44ab-929f-d8069e960533",
29      "description": null,
30      "state": "Active",
31      "name": "prod-11",
32      "status": "OK"
33    }
]
```



With version 1200, Scope Based Access Control and Associate an IPv6 subnet with an Ethernet are present
They were both introduced in OneView 4.00 (REST API Version 600).

API Version 1200	API Version 300
<pre>{ "type": "ethernet-networkV4", "uri": "/rest/ethernet-networks/89ad8bd5-718c-4f5c-be57-33acb49ff8d5", "category": "ethernet-networks", "eTag": "753c9ac6-75aa-4550-8c00-3ad179251c9e", "created": "2019-05-28T09:53:47.219Z", "modified": "2019-05-28T09:54:13.625Z", "scopesUri": "/rest/scopes/resources/rest/ethernet-networks/89ad8bd5-718c-4f5c-be57-33acb49ff8d5", "vlanId": 5, "subnetUri": "/rest/id-pools/ipv4/subnets/1d2213d7-c487-4166-93fd-e2d2c523f6ec", "IPv6SubnetUri": null, "connectionTemplateUri": "/rest/connection-templates/35cbd8ca-b73e-45b5-8858-77fd975aabac", "privateNetwork": false, "smartLink": false, "purpose": "Management", "ethernetNetworkType": "Tagged", "fabricUri": "/rest/fabrics/d3e64cd6-f176-44ab-929f-d8069e960533", "description": null, "state": "Active", "name": "Management", "status": "OK" },</pre>	<pre>{ "type": "ethernet-networkV300", "uri": "/rest/ethernet-networks/89ad8bd5-718c-4f5c-be57-33acb49ff8d5", "category": "ethernet-networks", "eTag": "753c9ac6-75aa-4550-8c00-3ad179251c9e", "created": "2019-05-28T09:53:47.219Z", "modified": "2019-05-28T09:54:13.625Z", "vlanId": 5, "subnetUri": "/rest/id-pools/ipv4/subnets/1d2213d7-c487-4166-93fd-e2d2c523f6ec", "connectionTemplateUri": "/rest/connection-templates/35cbd8ca-b73e-45b5-8858-77fd975aabac", "privateNetwork": false, "smartLink": false, "purpose": "Management", "ethernetNetworkType": "Tagged", "fabricUri": "/rest/fabrics/d3e64cd6-f176-44ab-929f-d8069e960533", "description": null, "state": "Active", "name": "Management", "status": "OK" },</pre>

So basically, any version below 600 will never provide/support any scopes and IPv6 associated subnet information.

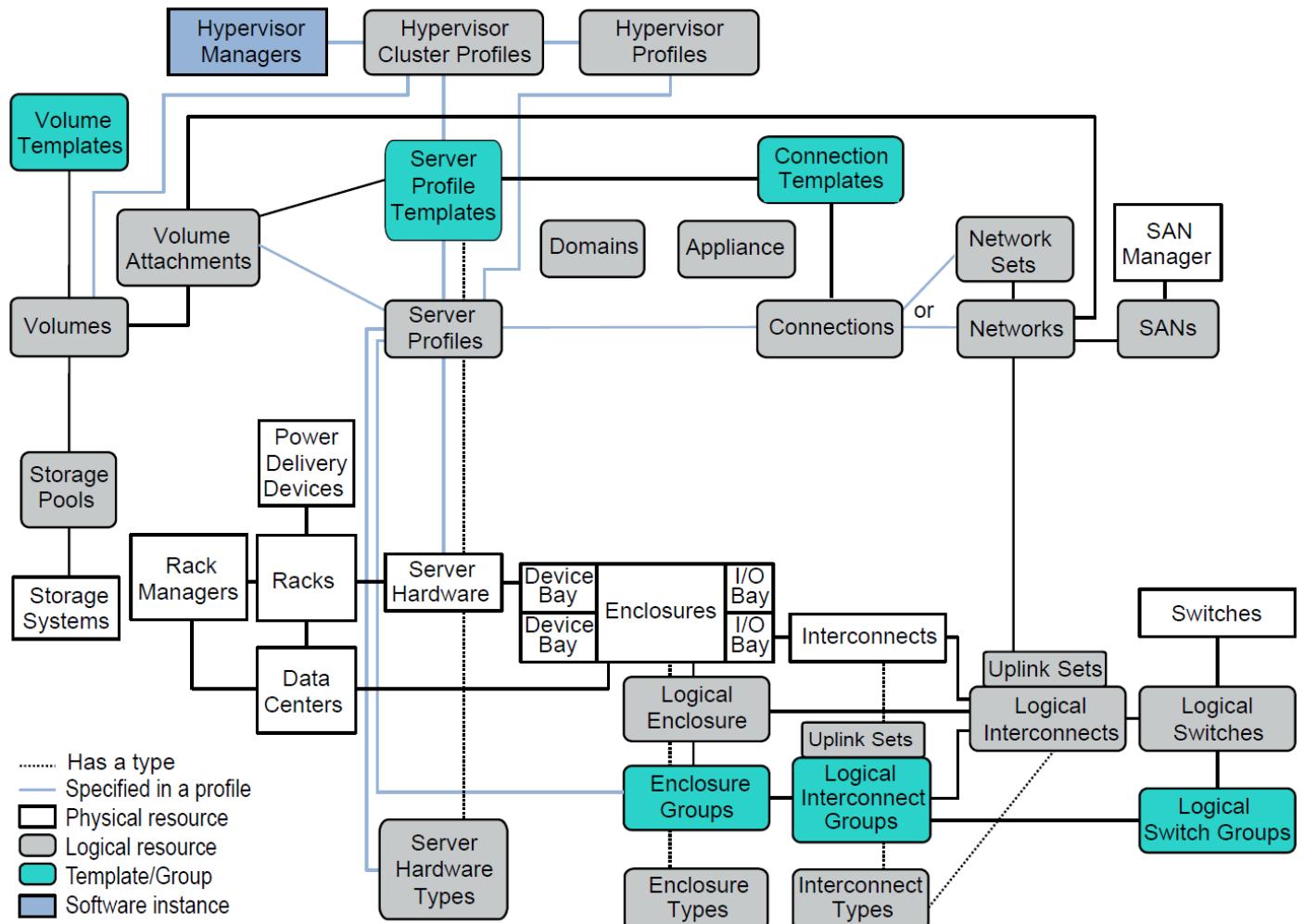


Lab 5: API Resource model

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.

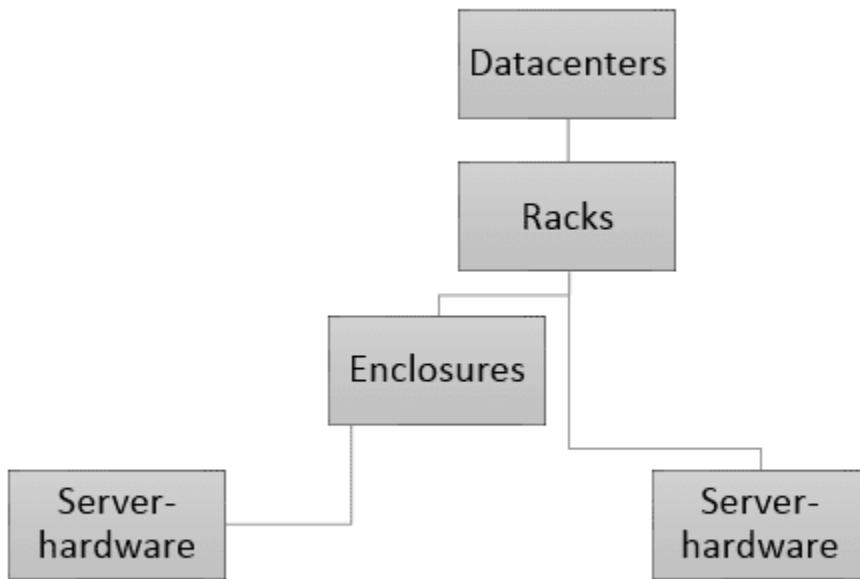
HPE OneView provides a fully connected resource model with associations or more precisely relationships between the HPE OneView resources. These associations are fundamental during script developments. The following figure summarizes some of the most frequently used resources and shows the relationships between them.





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

The following figure illustrates the objects we will examine during this process:



From this representation, we know that the topmost 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 `/rest/datacenters/schema` with our Postman utility.

The screenshot shows the Postman interface with a GET request to `https://composer.lj.lab/rest/datacenters/schema`. The Headers tab is active, displaying the following configuration:

Key	Value	Description
Content-Type	application/json	
x-api-version	1200	
auth	NDlxMDIzODg0NTU4DO4s46gggCKcY1s09M...	
Key	Value	Description

Temporary Headers (8) are also listed. The Body tab displays the JSON schema for a Datacenter object:

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

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

Note: To fold all regions, you can use **Alt + 0**



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 in a browser-based tool. The top navigation bar includes 'Body' (underlined), 'Cookies', 'Headers (12)', 'Test Results', 'Status: 200 OK', 'Time: 20ms', 'Size: 4.2 KB', and 'Save Response'. Below the status bar, there are tabs for 'Pretty', 'Raw', 'Preview', 'Visualize BETA', and 'JSON' (selected). The main area displays a large JSON object with line numbers on the left. The 'width' property is highlighted with a yellow box. The JSON structure is as follows:

```
1  {
2    "type": "object",
3    "id": "urn:jsonschema:com:hp:ci:mgmt:erm:model:Datacenter",
4    "properties": {
5      "width": {…},
6      "depth": {…},
7      "coolingCapacity": {…},
8      "costPerKilowattHour": {…},
9      "currency": {…},
10     "deratingType": {…},
11     "deratingPercentage": {…},
12     "coolingMultiplier": {…},
13     "contents": {…},
14     "category": {…},
15     "eTag": {…},
16     "created": {…},
17     "modified": {…},
18     "id": {…},
19     "uuid": {…},
20     "defaultPowerLineVoltage": {…},
21     "name": {…},
22     "uri": {…},
23     "state": {…},
24     "status": {…}
25   },
26   "description": "A data center object represents a facility used to house computer systems and associated components,
27 }
```

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

The screenshot shows a JSON response in a browser-based tool. The top navigation bar includes 'Body' (underlined), 'Cookies', 'Headers (12)', 'Test Results', 'Status: 200 OK', 'Time: 20ms', 'Size: 4.2 KB', and 'Save Response'. Below the status bar, there are tabs for 'Pretty', 'Raw', 'Preview', 'Visualize BETA', and 'JSON' (selected). The main area displays a large JSON object with line numbers on the left. The 'width' property is highlighted with a yellow box. The JSON structure is as follows:

```
1  {
2    "type": "object",
3    "id": "urn:jsonschema:com:hp:ci:mgmt:erm:model:Datacenter",
4    "properties": {
5      "width": {
6        "type": "integer",
7        "description": "The width in millimeters of the data center. This is associated with the PhysicalLocation x position.",
8        "maximum": "50000",
9        "minimum": "1000",
10       "required": "true"
11     }
12   }
13 }
```

We also can find a **required** flag, which is important, as required properties are mandatory when creating new objects with the **POST** method. Here *width* is mandatory when you create a datacenter object.

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

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

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 several 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     },  
82     "description": "The collection of physical resources (racks) in the data center and their  
83       position."  
84   },  
85   "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.

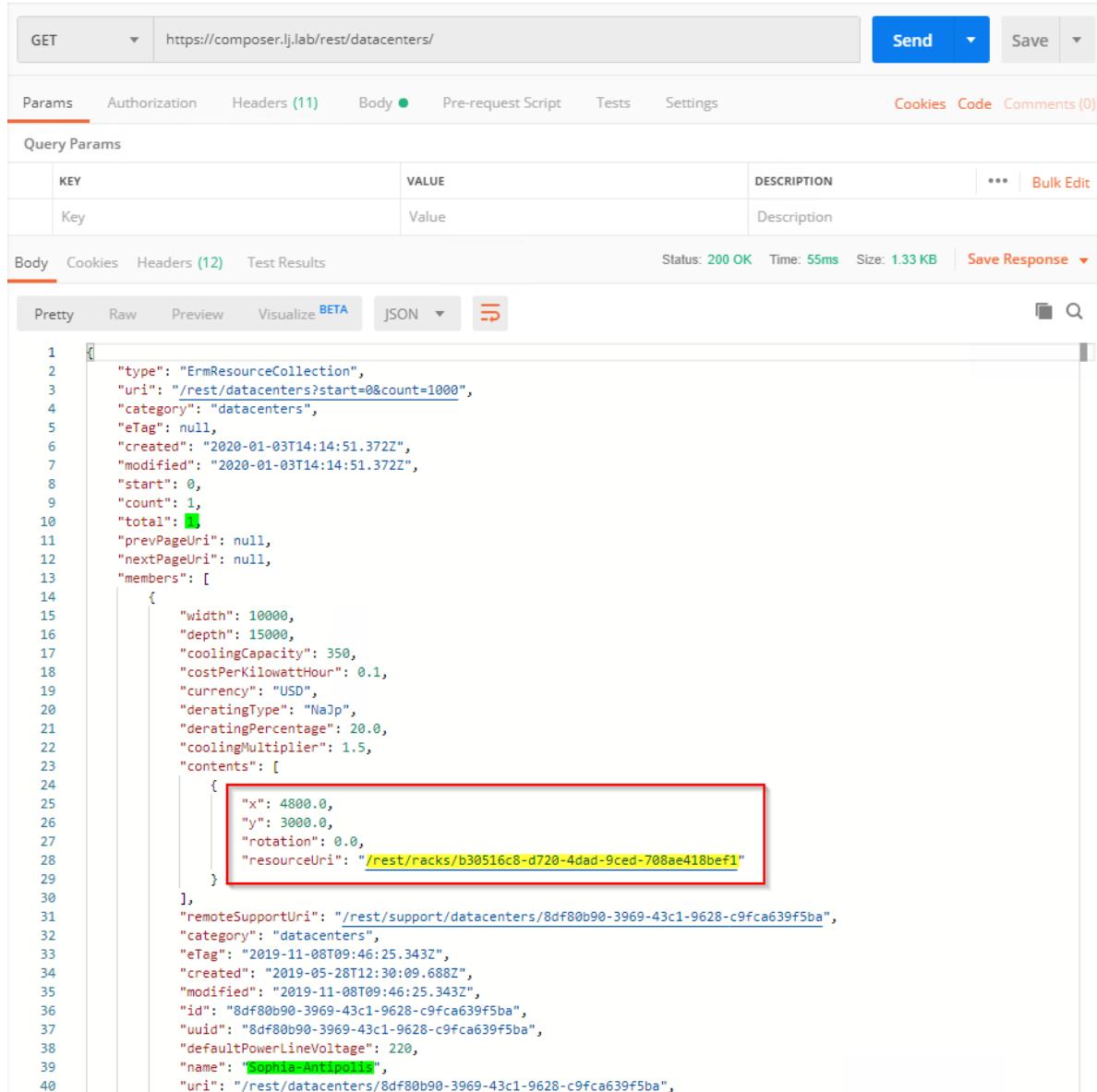
If you do a search for **GET /rest/datacenters** and scroll down to **Response Body**, you will find the same properties and descriptions of each object available in the response body, useful if you need to find a specific property for your script developments:

HPE OneView API Reference		
	Search	
API Reference		
About		
Security Model		
Response Codes		
Association Names		
Common Parameters		
Common Attributes		
What's New?		
SERVERS		
NETWORKING		
STORAGE		
FC-SANS		
FACILITIES		
Datacenters		
Power Devices		
Racks		
Unmanaged Devices		
EXTERNAL MANAGERS		
HYPERVISORS		
DATA SERVICES		
ACTIVITY		
SETTINGS		
SECURITY		
SEARCH		
REMOTE SUPPORT		
OS-DEPLOYMENT		
	Response Body	
		Datacenter List
	category	Identifies the resource type
		string read only
	count	The actual number of resources returned in the specified page
		integer
	created	Date and time when the resource was created
		timestamp read only
	Format	yyyy-MM-dd'T'HH:mm:ss.SSS'Z'
	Pattern	[1-2][0-9][0-9][0-9]-([0-1][0-9])-([0-3][0-9])T[0-2][0-9]:[0-5][0-9][0-9][0-9]Z
	eTag	Entity tag/version ID of the resource, the same value that is returned in the ETag header on a GET of the resource
		string read only
	members	The array of resources contained in the specified collection
	array of Datacenter	
	category	Identifies the resource type
		string read only
	contents	The collection of physical resources (racks) in the data center and their position.
	array of PhysicalLocation	
	resourceUri	The uri of the rack resource whose position is described.
		string
	rotation	The rotation (degrees) from 0-359 around the center of the resource.
		decimal
	Default	0
	x	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
	y	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



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>



The screenshot shows the Postman interface with a GET request to `https://composer.lj.lab/rest/datacenters`. The response is a JSON object representing a datacenter collection:

```
1  {
2      "type": "ErnResourceCollection",
3      "uri": "/rest/datacenters?start=0&count=1000",
4      "category": "datacenters",
5      "eTag": null,
6      "created": "2020-01-03T14:14:51.372Z",
7      "modified": "2020-01-03T14:14:51.372Z",
8      "start": 0,
9      "count": 1,
10     "total": 1,
11     "prevPageUri": null,
12     "nextPageUri": null,
13     "members": [
14         {
15             "width": 10000,
16             "depth": 15000,
17             "coolingCapacity": 350,
18             "costPerKilowattHour": 0.1,
19             "currency": "USD",
20             "deratingType": "NaJp",
21             "deratingPercentage": 20.0,
22             "coolingMultiplier": 1.5,
23             "contents": [
24                 {
25                     "x": 4800.0,
26                     "y": 3000.0,
27                     "rotation": 0.0,
28                     "resourceUri": "/rest/racks/b30516c8-d720-4dad-9ced-708ae418bef1"
29                 }
30             ],
31             "remoteSupportUri": "/rest/support/datacenters/8df80b90-3969-43c1-9628-c9fca639f5ba",
32             "category": "datacenters",
33             "eTag": "2019-11-08T09:46:25.343Z",
34             "created": "2019-05-28T12:30:09.688Z",
35             "modified": "2019-11-08T09:46:25.343Z",
36             "id": "8df80b90-3969-43c1-9628-c9fca639f5ba",
37             "uuid": "8df80b90-3969-43c1-9628-c9fca639f5ba",
38             "defaultPowerLineVoltage": 220,
39             "name": "Sophia-Antipolis",
40             "uri": "/rest/datacenters/8df80b90-3969-43c1-9628-c9fca639f5ba",
41         }
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 URI: `/rest/racks/b30516c8-d720-4dad-9ced-708ae418bef1"`



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 b30516c8-d720-4dad-9ced-708ae418bef1

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:

- Request Method:** GET
- URL:** https://composer.lj.lab/rest/racks/schema
- Headers (10):**
 - Content-Type: application/json
 - x-api-version: 1200
 - auth: NDIxMDIzODg0NTU4DO4s46gggCKcY1s09MhZo...
 - Key: Value
- Body:** A JSON schema for a Rack object, shown in Pretty mode:

```
1 {  
2   "type": "object",  
3   "id": "urn:jsonschema:com:hp:ci:mgmt:erm:model:Rack",  
4   "properties": {  
5     "uHeight": {…},  
6     "width": {…},  
7     "height": {…},  
8     "depth": {…},  
9     "thermalLimit": {…},  
10    "partNumber": {…},  
11    "serialNumber": {…},  
12    "rackMounts": {…},  
13    "category": {…},  
14    "eTag": {…},  
15    "created": {…}  
16  }  
17}
```
- Status:** 200 OK
- Time:** 22ms
- Size:** 3.26 KB
- Save Response:** Available

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.

```
42 },
43 },
44 "serialNumber": {...},
45 },
46 "rackMounts": {
47     "type": "array",
48     "items": {
49         "type": "object",
50         "id": "urn:jsonschema:com:hp:ci:mgmt:erm:model:RackMount",
51         "properties": {
52             "mountUri": {...},
53             "topUSlot": {...},
54             "uHeight": {...},
55             "relativeOrder": {...},
56             "location": {...}
57         }
58     },
59     "description": "References to the resources contained in the rack."
60 },
61 },
62 },
63 },
64 },
65 },
66 },
67 },
68 },
69 },
70 },
71 },
72 },
73 },
74 },
75 },
76 },
77 },
78 },
79 },
80 },
81 },
82 },
83 },
84 },
85 },
86 },
87 },
```

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



```
13 "members": [
14     {
15         "uHeight": 42,
16         "width": 600,
17         "height": 2004,
18         "depth": 1075,
19         "thermalLimit": 10000,
20         "partNumber": "AF046A",
21         "serialNumber": "AABB1122CCDD",
22         "rackMounts": [
23             {
24                 "mountUri": "/rest/unmanaged-devices/e45aca3a-5d6d-4acd-8650-4609e9af805",
25                 "topUSlot": 40,
26                 "uHeight": 1,
27                 "relativeOrder": 0,
28                 "location": "CenterFront"
29             },
23             {
24                 "mountUri": "/rest/unmanaged-devices/4083bc3f-a826-4c80-b6f2-c9893e30fa70",
25                 "topUSlot": 37,
26                 "uHeight": 1,
27                 "relativeOrder": 0,
28                 "location": "CenterFront"
29             },
23             {
24                 "mountUri": "/rest/enclosures/000000CN7515049",
25                 "topUSlot": 20,
26                 "uHeight": 10,
27                 "relativeOrder": -1,
28                 "location": "CenterFront"
29             },
23             {
24                 ...
25             },
23             {
24                 ...
25             },
23             {
24                 ...
25             },
23             {
24                 ...
25             },
23             {
24                 ...
25             },
23             {
24                 ...
25             },
23             {
24                 ...
25             },
23             {
24                 ...
25             }
26         ],
27         "model": "",
28         "category": "racks",
29         "eTag": "2019-11-08T09:42:07.087Z",
30         "created": "2019-05-28T10:09:18.680Z",
31         "modified": "2019-11-08T09:42:07.087Z",
32         "id": "b30516c8-d720-4dad-9ced-708ae418bef1",
33         "uuid": "b30516c8-d720-4dad-9ced-708ae418bef1",
34         "name": "Rack-Synergy",
35         "uri": "/rest/racks/b30516c8-d720-4dad-9ced-708ae418bef1".
```

From that query, we discover that our rack is named “Rack-Synergy”, its part number is AF046A and in this rack we have several enclosures, servers and unmanaged components (switches). We have, with the *mountUri* property, a direct access to the URI of each component 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  {
2      "type": "EnclosureV8",
3      "uri": "/rest/enclosures/000000CN7515049L",
4      "category": "enclosures",
5      "eTag": "2020-01-03T09:13:18.423Z",
6      "created": "2019-05-28T09:09:23.620Z",
7      "modified": "2020-01-03T09:13:18.423Z",
8      "refreshState": "NotRefreshing",
9      "stateReason": "None",
10     "enclosureType": "SY12000",
11     "enclosureTypeUri": "/rest/enclosure-types/SY12000",
12     "enclosureModel": "Synergy 12000 Frame",
13     "uid": "000000CN7515049L",
14     "serialNumber": "CN7515049L",
15     "partNumber": "000000-010",
16     "reconfigurationState": "NotReapplyingConfiguration",
17     "uidState": "Off",
18     "licensingIntent": "NotApplicable",
19     "deviceBayCount": 12,
20     "deviceBays": [...],
289 ],
290     "interconnectBayCount": 6,
291     "interconnectBays": [...],
292 ],
293     "fanBayCount": 10,
294     "fanBays": [...],
295 ],
296     "powerSupplyBayCount": 6,
297     "powerSupplyBays": [...],
298 ],
299     "enclosureGroupUri": "/rest/enclosure-groups/8ffcfec5-fbb2-46a2-a843-b2e434066e0d",
300     "fwBaselineUri": "/rest/firmware-drivers/HPE_Synergy_Custom_SPP_2019_09_20190926_Z7550-96770",
301     "fwBaselineName": "HPE Synergy Custom SPP 201909 2019 09 26, 2019.09.26.00",
302     "isFwManaged": true,
303     "forceInstallFirmware": false,
304     "logicalEnclosureUri": "/rest/logical-enclosures/f3550ea7-ebed-491a-87e2-f86d27a02305",
305     "managerBays": [...],
306 ],
307     "supportState": "Disabled",
308     "supportDataCollectionState": null,
309     "supportDataCollectionType": null,
310     "supportDataCollectionsUri": "/rest/support/data-collections?deviceID=000000CN7515049L&category=enclosures",
311     "remoteSupportUri": "/rest/support/enclosures/000000CN7515049L",
312     "remoteSupportSettings": {...},
313 },
314     "crossBars": [],
315     "partitions": [],
316     "scopesUri": "/rest/scopes/resources/rest/enclosures/000000CN7515049L",
317     "status": "OK",
318     "name": "Frame2",
319     "state": "Configured",
320     "description": null,
321     "frameLinkModuleDomain": "local".
```

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     "powerAllocationWatts": null.

```

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

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



Associations

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

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

```
1117 },
1118 {
1119     "type": "ServerProfileTemplateV7",
1120     "uri": "/rest/server-profile-templates/9b42bfb9-8b79-41fb-a78d-6c383bbcd0c2",
1121     "name": "RHEL7.3 deployment with Streamer",
1122     "description": "Server Profile Template for HPE Synergy 480 Gen9 Compute Module using the Image Streamer",
1123     "serverProfileDescription": "root / P@ssw0rd - demopaq / HPEinvent",
1124     "serverHardwareTypeUri": "/rest/server-hardware-types/DC832854-EE28-46DE-9768-4F35F1C7A92E",
1125     "enclosureGroupUri": "/rest/enclosure-groups/8ffcfec5-fbb2-46a2-a843-b2e434066e0d",
1126     "affinity": "Bay",
1127     "hideUnusedFlexNics": true,
1128     "macType": "Virtual",
1129     "wwnType": "Virtual",
1130     "serialNumberType": "Virtual",
1131     "iscsiInitiatorNameType": "AutoGenerated",
1132     "osDeploymentSettings": {
1133         "complianceControl": "Checked",
1134         "osDeploymentPlanUri": "/rest/os-deployment-plans/aaed2ac3-dcf9-4740-9d1b-6414b6cefd9d",
1135         "osCustomAttributes": [
1136             {
1137                 "name": "HostName",
1138                 "value": "{profile}",
1139                 "constraints": "{}\\\"\\\"{}",
1140                 "type": "hostname"
1141             },
1142             {
1143                 "name": "Team0NIC1.networkuri",
1144                 "value": "/rest/ethernet-networks/89ad8bd5-718c-4f5c-be57-33acb49ff8d5",
1145                 "constraints": null,
1146                 "type": null
1147             },
1148             {
1149                 "name": "Team1NIC2.networkuri",
1150                 "value": "/rest/ethernet-networks/718179dd-f0f8-409d-a4ab-a19e9f08e4a7",
1151                 "constraints": null,
1152                 "type": null
1153             },
1154             {
1155                 "name": "NICTeam0Name",
1156                 "value": "team0",
1157                 "constraints": "{}",
1158                 "type": "string"
1159             }
1160         ]
1161     }
1162 }
```



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. On the left, a sidebar lists "Server Profile Templates" with 10 matches, including options like "Create server profile template", "Name", and "RHEL7.3 deployment with Streamer" (which is highlighted). On the right, the main pane displays the details for the selected "RHEL7.3 deployment with Streamer" template. The "General" tab is active, showing the following configuration:

Description	Value
Server Profile Template for HPE Synergy 480 Gen9 Compute Module using the Image Streamer	
Server profile description	root / P@ssw0rd - demopaq / HPEinvent
Server hardware type	SY 480 Gen9 1
Enclosure group	3 frame EG
Affinity	Device bay
Connections	6
SAN volume attachments	managed manually
Logical drives	0
Logical JBODs	0
Firmware baseline	managed manually
BIOS	managed manually
iLO Settings	managed manually

The "Server Profiles" section shows a circular icon with the number "2" and the status "Inconsistent".

Additionally, enquiring a list of server profiles managed by a server profile template is 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:

HPE OneView API Reference

Association Names

Parent	Child	Association name
connections	interconnects	CONNECTION_TO_INTERCONNECT
datacenters	racks	DATACENTER_TO_PHYSICAL_OBJECT
drive-enclosures	drive-enclosures/drive-bays	DRIVE_ENCLOSURE_TO_DRIVE_BAY_ASSOC
	sas-logical-interconnects	DRIVE_ENCLOSURE_TO_SAS_LOGICAL_INTERCONNECT
drive-enclosures/drive-bays	drive-enclosures/drive-bays/drives	DRIVE_BAY_TO_DRIVE_ASSOC
enclosures	drive-enclosures	ENCLOSURE_TO_BLADE
	enclosures/device-bays	ENCLOSURE_TO_DEVICE_BAY
	interconnects	ENCLOSURE_TO_INTERCONNECT
	logical-interconnects	ENCLOSURE_TO_LOGICAL_INTERCONNECT
	sas-interconnects	ENCLOSURE_TO_INTERCONNECT
	sas-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_GI
	sas-logical-interconnect-groups	ENCLOSURE_GROUP_TO_LOGICAL_INTERCONNECT_GI
	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-	drive-enclosures	DEVICE_BAY_TO_SERVER_HARDWARE

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`

HPE OneView API Reference

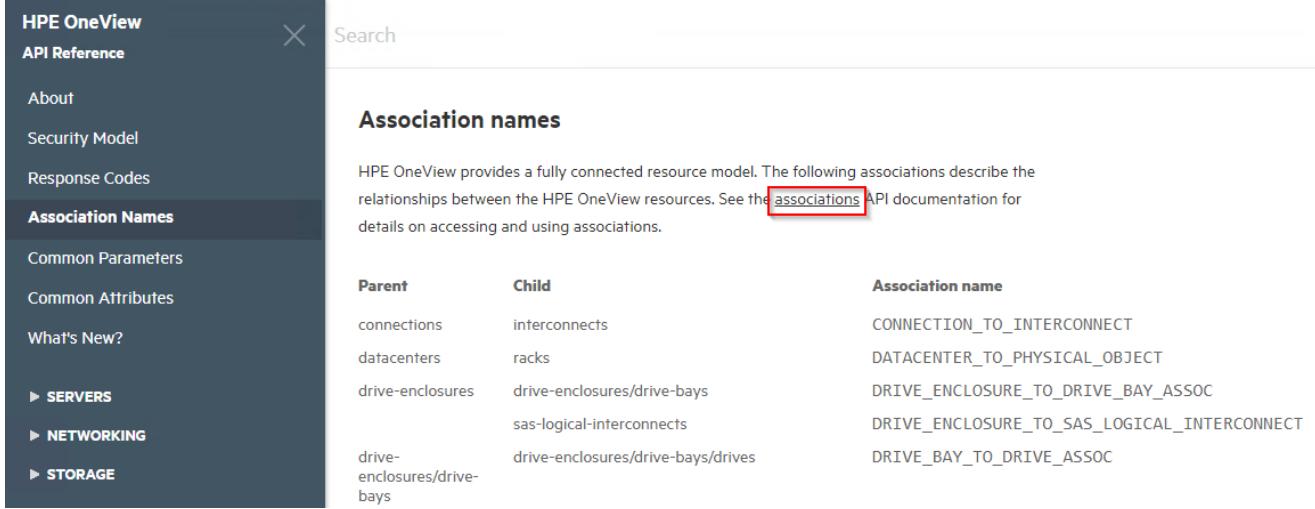
Association Names

server-profile-templates	server-profiles	<code>server_profile_template_to_server_profiles</code>
--------------------------	-----------------	---



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**.

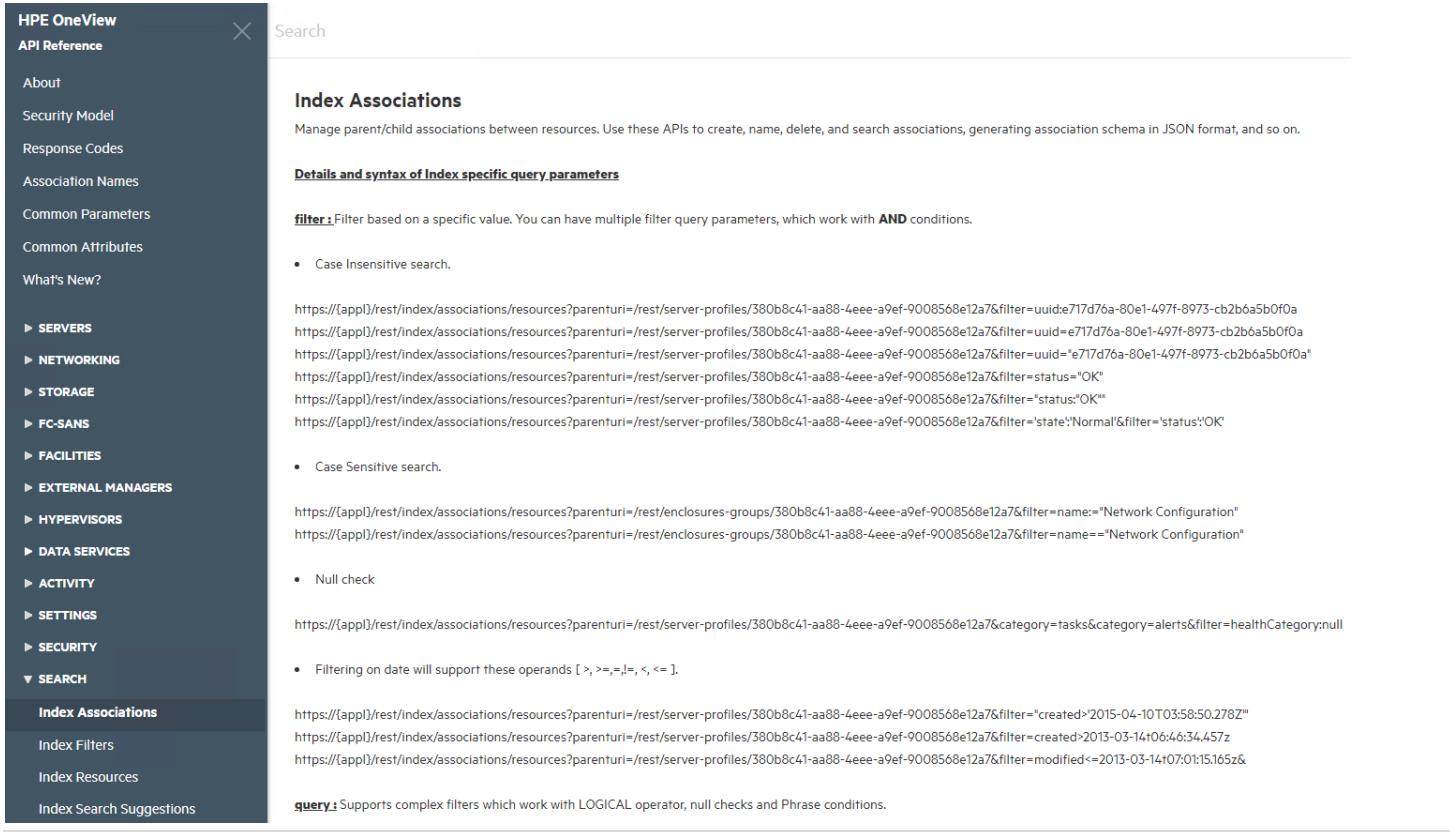


Association names

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.

Parent	Child	Association name
connections	interconnects	CONNECTION_TO_INTERCONNECT
datacenters	racks	DATACENTER_TO_PHYSICAL_OBJECT
drive-enclosures	drive-enclosures/drive-bays	DRIVE_ENCLOSURE_TO_DRIVE_BAY_ASSOC
	sas-logical-interconnects	DRIVE_ENCLOSURE_TO_SAS_LOGICAL_INTERCONNECT
drive-enclosures/drive-bays	drive-enclosures/drive-bays/drives	DRIVE_BAY_TO_DRIVE_ASSOC

In this page, we can find several examples of API request to create, delete, name and search associations.



Index Associations

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.

Details and syntax of index specific query parameters

filter: Filter based on a specific value. You can have multiple filter query parameters, which work with **AND** conditions.

- Case Insensitive search.

```
https://[app]/rest/index/associations/resources?parenturi=/rest/server-profiles/380b8c41-aa88-4eee-a9ef-9008568e12a7&filter=uuid=e71d76a-80e1-497f-8973-cb2b6a5b0f0a
https://[app]/rest/index/associations/resources?parenturi=/rest/server-profiles/380b8c41-aa88-4eee-a9ef-9008568e12a7&filter=uuid=e71d76a-80e1-497f-8973-cb2b6a5b0f0a
https://[app]/rest/index/associations/resources?parenturi=/rest/server-profiles/380b8c41-aa88-4eee-a9ef-9008568e12a7&filter=uuid=e71d76a-80e1-497f-8973-cb2b6a5b0f0a
https://[app]/rest/index/associations/resources?parenturi=/rest/server-profiles/380b8c41-aa88-4eee-a9ef-9008568e12a7&filter=status="OK"
https://[app]/rest/index/associations/resources?parenturi=/rest/server-profiles/380b8c41-aa88-4eee-a9ef-9008568e12a7&filter="status:"OK"
https://[app]/rest/index/associations/resources?parenturi=/rest/server-profiles/380b8c41-aa88-4eee-a9ef-9008568e12a7&filter="state:'Normal"&filter="status:'OK'
```

- Case Sensitive search.

```
https://[app]/rest/index/associations/resources?parenturi=/rest/enclosures-groups/380b8c41-aa88-4eee-a9ef-9008568e12a7&filter=name=="Network Configuration"
https://[app]/rest/index/associations/resources?parenturi=/rest/enclosures-groups/380b8c41-aa88-4eee-a9ef-9008568e12a7&filter=name=="Network Configuration"
```

- Null check

```
https://[app]/rest/index/associations/resources?parenturi=/rest/server-profiles/380b8c41-aa88-4eee-a9ef-9008568e12a7&category=tasks&category=alerts&filter=healthCategory=null
```

- Filtering on date will support these operands [>, >=, !=, <, <=].

```
https://[app]/rest/index/associations/resources?parenturi=/rest/server-profiles/380b8c41-aa88-4eee-a9ef-9008568e12a7&filter="created>2015-04-10T03:58:50.278Z"
https://[app]/rest/index/associations/resources?parenturi=/rest/server-profiles/380b8c41-aa88-4eee-a9ef-9008568e12a7&filter="created>2013-03-14T06:46:34.457Z
https://[app]/rest/index/associations/resources?parenturi=/rest/server-profiles/380b8c41-aa88-4eee-a9ef-9008568e12a7&filter="modified<=2013-03-14T07:01:15.165Z&
```

query: Supports complex filters which work with LOGICAL operator, null checks and Phrase conditions.



The query parameters to get the list of associations that match the requested parameters use the following syntax:
`https://{{appl}}/rest/index/associations?parenturi={{resource URI}}&{{parameters}}`

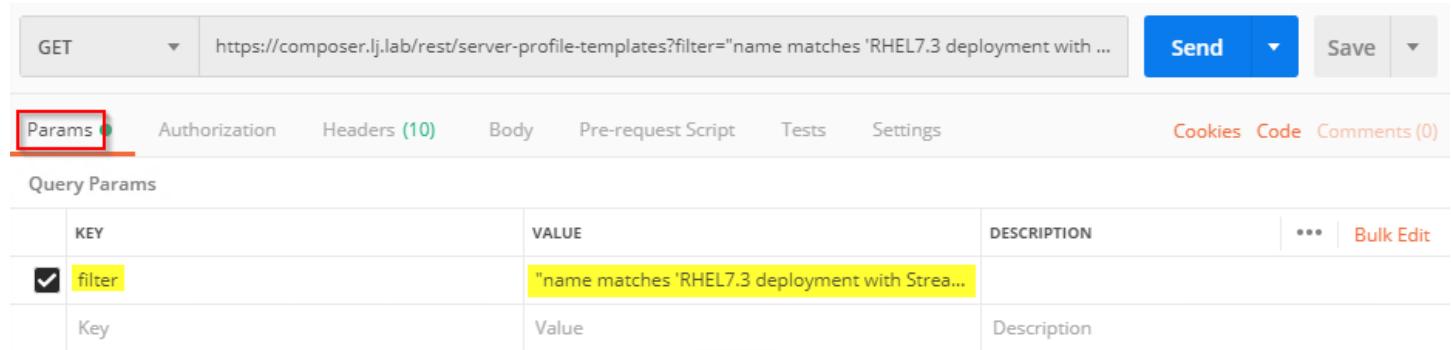
The query parameters can be used with `parenturi/childuri/name` filtering.

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

Field	Value
URL	<code>https://composer.lj.lab/rest/server-profile-templates</code>
Params Key	<code>filter</code>
Params Value	<code>"name matches 'RHEL7.3 deployment with Streamer'"</code>
Verb	<code>GET</code>
Header	<code>Auth : your sessionID</code> <code>X-API-Version:1200</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 'RHEL7.3 deployment with Streamer'"`



GET https://composer.lj.lab/rest/server-profile-templates?filter="name matches 'RHEL7.3 deployment with Streamer'" Send Save

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies Code Comments (0)

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> filter	"name matches 'RHEL7.3 deployment with Streamer'"			
Key	Value	Description		

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



Body Cookies Headers (12) Test Results Status: 200 OK Time: 28ms Size: 7.36 KB Save Response ▾

Pretty Raw Preview Visualize **BETA** JSON

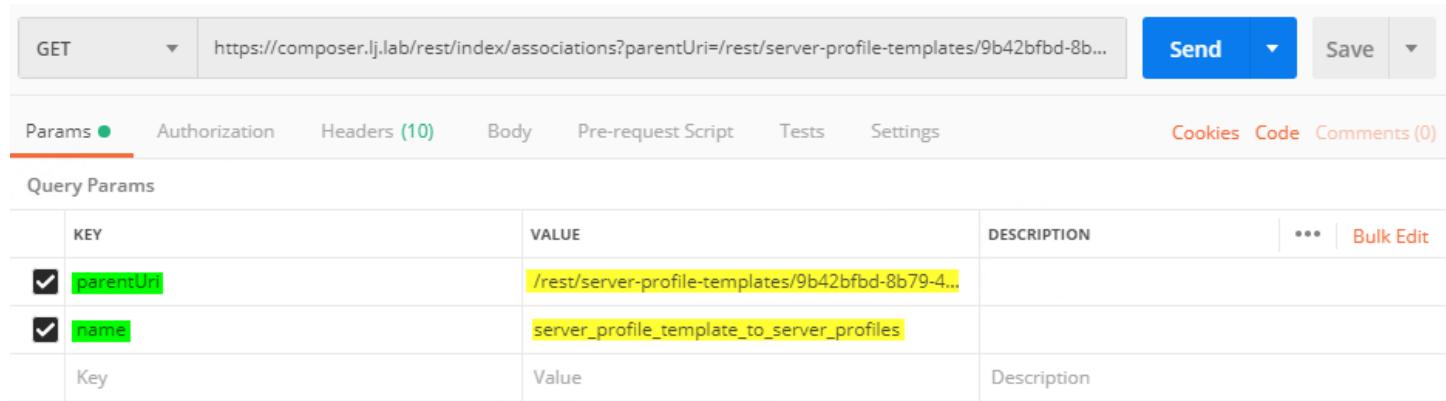
```
1 {"type": "ServerProfileTemplateListV7",
2  "uri": "/rest/server-profile-templates?filter=%22name%20matches%20%27RHEL7.3%20deployment%20with%20Streamer%27%22&
3   count=64",
4  "category": "server-profile-templates",
5  "eTag": "2020-01-03T15:11:47.621Z",
6  "created": "2020-01-03T15:11:47.621Z",
7  "modified": "2020-01-03T15:11:47.621Z",
8  "start": 0,
9  "count": 1,
10 "total": 1,
11 "prevPageUri": null,
12 "nextPageUri": null,
13 "members": [
14   {
15     "type": "ServerProfileTemplateV7",
16     "uri": "/rest/server-profile-templates/9b42bfbd-8b79-41fb-a78d-6c383bbcd0c2",
17     "name": "RHEL7.3 deployment with Streamer",
18     "description": "Server Profile Template for HPE Synergy 480 Gen9 Compute Module using the Image Streamer",
19     "serverProfileDescription": "root / P@ssw0rd - demopaq / HPEinvent",
20     "serverHardwareTypeUri": "/rest/server-hardware-types/DC832854-EE28-46DE-9768-4F35F1C7A92E",
21     "enclosureGroupUri": "/rest/enclosure-groups/8ffcfec5-fbb2-46a2-a843-b2e434066e0d",
```

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	<code>https://composer.lj.lab/rest/index/associations</code>
Parameter key	<code>parentUri</code>
Parameter value	<code>/rest/server-profile-templates/9b42bfb9-8b79-41fb-a78d-6c383bbcd0c2</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:1200</code>



The screenshot shows the Postman interface with a GET request. The URL is `https://composer.lj.lab/rest/index/associations?parentUri=/rest/server-profile-templates/9b42bfb9-8b79-41fb-a78d-6c383bbcd0c2&name=server_profile_template_to_server_profiles`. The 'Params' tab is active, showing two selected parameters: `parentUri` and `name`. Other tabs include Authorization, Headers (10), Body, Pre-request Script, Tests, Settings, Cookies, Code, and Comments (0).

Notice that using those parameters modify the URL to

`https://composer.lj.lab/rest/index/associations?parentUri=/rest/server-profile-templates/9b42bfb9-8b79-41fb-a78d-6c383bbcd0c2&name=server_profile_template_to_server_profiles`



This is what the response body looks like:

The screenshot shows a Postman interface with the following details:

- Body tab selected.
- Headers (12) tab selected.
- Status: 200 OK, Time: 29ms, Size: 1.08 KB, Save Response button.
- JSON dropdown selected.
- Pretty, Raw, Preview, Visualize BETA buttons.
- Search and refresh icons.
- JSON code:

```
1 {
2     "type": "AssociationsV200",
3     "uri": "/rest/index/associations?name=server_profile_template_to_server_profiles&start=0&count=50&parentUri=/rest/
4         server-profile-templates/9b42bfbd-8b79-41fb-a78d-6c383bbcd0c2",
5     "category": null,
6     "eTag": null,
7     "created": null,
8     "modified": null,
9     "start": 0,
10    "count": 2,
11    "total": 2,
12    "prevPageUri": null,
13    "nextPageUri": null,
14    "members": [
15        {
16            "parentUri": "/rest/server-profile-templates/9b42bfbd-8b79-41fb-a78d-6c383bbcd0c2",
17            "childUri": "/rest/server-profiles/155d77c5-2ec5-4901-991b-a97a9513f67c",
18            "name": "server_profile_template_to_server_profiles"
19        },
20        {
21            "parentUri": "/rest/server-profile-templates/9b42bfbd-8b79-41fb-a78d-6c383bbcd0c2",
22            "childUri": "/rest/server-profiles/decd5e30-8b51-42a6-9da4-3563371d9e14",
23            "name": "server_profile_template_to_server_profiles"
24        }
25    ]
}
```

The response shows that the Server Profile Template `/rest/server-profile-templates/9b42bfbd-8b79-41fb-a78d-6c383bbcd0c2` has two Server profiles `/rest/server-profiles/155d77c5-2ec5-4901-991b-a97a9513f67c` and `/rest/server-profiles/decd5e30-8b51-42a6-9da4-3563371d9e14` associated with it.

To access to the server profile information, you can simply click on one of the `childUri` link:

The screenshot shows a Postman interface with the following details:

- Body tab selected.
- Headers (12) tab selected.
- Status: 200 OK, Time: 29ms, Size: 1.08 KB, Save Response button.
- JSON dropdown selected.
- Pretty, Raw, Preview, Visualize BETA buttons.
- Search and refresh icons.
- JSON code:

```
8 {
9     "start": 0,
10    "count": 2,
11    "total": 2,
12    "prevPageUri": null,
13    "nextPageUri": null,
14    "members": [
15        {
16            "parentUri": "/rest/server-profile-templates/9b42bfbd-8b79-41fb-a78d-6c383bbcd0c2",
17            "childUri": "/rest/server-profiles/155d77c5-2ec5-4901-991b-a97a9513f67c",
18            "name": "server_profile_template_to_server_profiles"
19        },
20        {
21            "parentUri": "/rest/server-profile-templates/9b42bfbd-8b79-41fb-a78d-6c383bbcd0c2",
22            "childUri": "/rest/server-profiles/decd5e30-8b51-42a6-9da4-3563371d9e14",
23            "name": "server_profile_template_to_server_profiles"
24        }
25    ]
}
```



Then click **Send**

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** <https://composer.lj.lab/rest/server-profiles/155d77c5-2ec5-4901-991b-a97a9513f67c>
- Buttons:** Send (highlighted in blue), Save
- Params Tab:** Active, showing a single query parameter "Key" with value "Value".
- Headers Tab:** (10) items listed.
- Body Tab:** Active, showing the JSON response.
- Response Headers:** Status: 200 OK, Time: 31ms, Size: 7.42 KB, Save Response
- JSON Response:**

```
1 {
2     "type": "ServerProfileV11",
3     "uri": "/rest/server-profiles/155d77c5-2ec5-4901-991b-a97a9513f67c",
4     "name": "rh-1",
5     "description": "",
6     "serialNumber": "VCGL952027",
7     "uuid": "155d77c5-2ec5-4901-991b-a97a9513f67c",
8     "iscsiInitiatorName": "iqn.2015-02.com.hpe:oneview-vcg1952027",
9     "iscsiInitiatorNameType": "AutoGenerated",
10    "serverProfileTemplateUri": "/rest/server-profile-templates/9b42bfb9-8b79-41fb-a78d-6c383bbcd0c2",
11    "templateCompliance": "NonCompliant",
12    "serverHardwareUri": "/rest/server-hardware/36343537-3338-4E43-3736-30313042355A",
13    "serverHardwareTypeUri": "/rest/server-hardware-types/DC832854-EE28-46DE-9768-4F35F1C7A92E",
14    "enclosureGroupUri": "/rest/enclosure-groups/8ffcfec5-fbb2-46a2-a843-b2e434066e0d",
15    "enclosureUri": "/rest/enclosures/000000CN7516060D",
16    "enclosureBay": 2,
17    "affinity": "Bay",
18    "associatedServer": null,
19    "hideUnusedFlexNics": true,
20    "firmware": {
21        "firmwareBaselineUri": null,
22        "manageFirmware": false,
23        "forceInstallFirmware": false,
24        "firmwareInstallType": null,
25        "firmwareScheduleDateTime": null,
26        "firmwareActivationType": null,
27        "reapplyState": "NotApplying",
28        "consistencyState": "Consistent"
29    },
30    "macType": "Virtual",
31    "wwnType": "Virtual",
32    "serialNumberType": "Virtual",
33    "category": "server-profiles",
34 }
```

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 disabled and with an iSCSI purpose?

Hint: RTFM, everything can be found in the documentation!

Answer: on next page.

Exercise 2

How can you configure a remote SYSLOG server in HPE OneView GUI?

Note: Remote Syslog is different than Audit Logs Forwarding which enables HPE OneView to forward audit logs to remote Security Information and Event Management (SIEM) systems. Remote Syslog provides the ability to receive the logs from supported devices managed by OneView like iLOs, interconnect modules, etc.

Hint: 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 since 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=false&filter=purpose=Management

Send Save

Params (1) Authorization Headers (10) Body Pre-request Script Tests Settings Cookies Code Comments (0)

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> filter	smartLink=false			
<input checked="" type="checkbox"/> filter	purpose=Management			
Key	Value	Description		

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 allows you to manage up to 75 HPE OneView or HPE Synergy instances, and 150 HPE Hyper Converged 380 instances across data centers.

It helps your IT staff troubleshoot alerts and view core inventory data from one place and make more informed, faster decisions with better infrastructure visibility. With single sign-on, you are one click away from powerful, device-level lifecycle management.

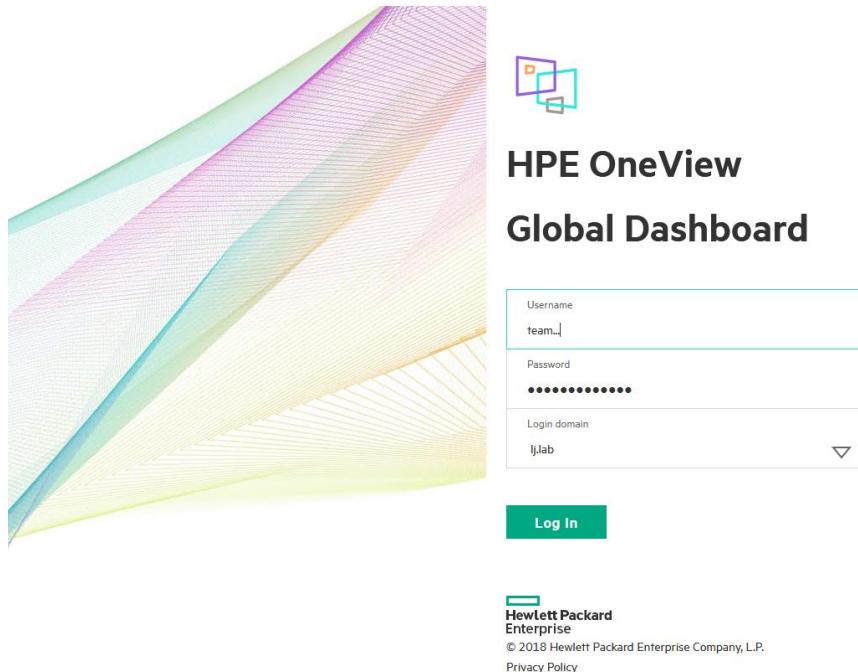
At no additional cost, you can simply download and run HPE OneView Global Dashboard alongside HPE OneView, allowing you to see your HPE Synergy, HPE BladeSystem c-Class, HPE ProLiant DL, HPE Hyper Converged 380, HPE SimpliVity 380, HPE Apollo, HPE ML350 Gen9 servers and HPE ML Gen10 servers, HPE Composable Cloud for DL systems, and HPE Superdome FLEX systems as one infrastructure.

Since version 1.6, released in October 2018, a RESTful API is available with HPE OneView Global Dashboard. A RESTful API means that you can query all resources that have been added to Global Dashboard to access inventory, health status, and alerting information via command lines, and even scripting 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

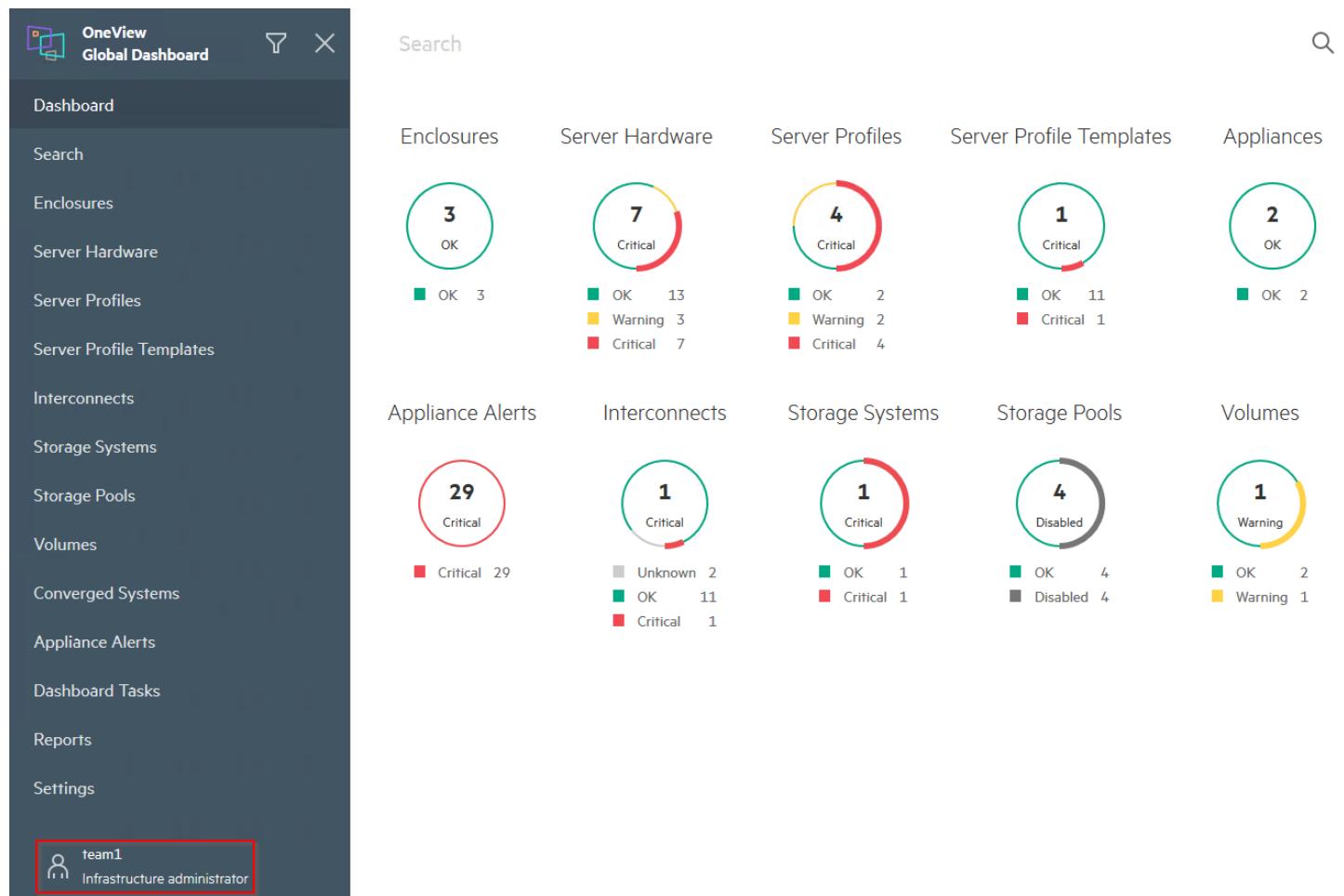




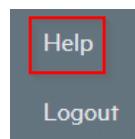
Hewlett Packard Enterprise

TSS Paris 2020

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



Select **Help**





Then click on **REST API Reference**

The dashboard shows the following summary:

Category	Status	Count
Enclosures	OK	3
Server Hardware	Critical	1
Server Profile Templates	OK	24
	Warning	2
	Critical	1

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.

HPE OneView Global Dashboard REST API (v2)

Download OpenAPI specification: [Download](#)

Product version 1.90.00

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

REQUEST SAMPLES

```
POST /rest/appliances
```

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



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

Click on **Login Sessions**

The screenshot shows the HPE OneView Global Dashboard REST API (v2) interface. On the left, there is a sidebar with a search bar and a list of resources: APPLIANCES, CERTIFICATES, CONVERGED SYSTEMS, ENCLOSURES, GROUPS, INTERCONNECTS, LOGIN SESSIONS, and MANAGED SANS. The 'LOGIN SESSIONS' item is highlighted with a red border. The main content area displays the 'APPLIANCES' section, which includes the title 'HPE OneView Global Dashboard REST API (v2)', a link to 'Download OpenAPI specification' (with a 'Download' button), the 'Product version 1.90.00', and a sub-section titled 'Appliances' with the description 'Operations on appliances'.

Resource	Description
APPLIANCES	Operations on appliances
CERTIFICATES	
CONVERGED SYSTEMS	
ENCLOSURES	
GROUPS	
INTERCONNECTS	
LOGIN SESSIONS	Operations on login sessions
MANAGED SANS	



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.

The screenshot shows the HPE OneView Global Dashboard API documentation for the 'Login Sessions' resource. The left sidebar lists various resources, and the main content area shows the details for 'Login Sessions'. A red box highlights the 'Create a session' section under the 'PARAMETERS' tab, which contains two required header parameters: 'Content-Type' (string) and 'X-Api-Version' (integer). A red circle labeled '1' is on the sidebar, and another red circle labeled '2' is on the main content area. To the right, a dark-paneled window displays 'REQUEST SAMPLES' and 'RESPONSE SAMPLES' for the POST /rest/login-sessions/ endpoint. A red circle labeled '3' is on this panel.

APPLIANCES

CERTIFICATES

CONVERGED SYSTEMS

ENCLOSURES

GROUPS

INTERCONNECTS

LOGIN SESSIONS

MANAGED SANS

NETWORK INTERFACES

RESOURCE ALERTS

SAN MANAGERS

SERVER FIRMWARE

SERVER HARDWARE

SERVER PROFILES

SERVER PROFILE TEMPLATES

STORAGE POOLS

Search

Login Sessions

Operations on login sessions

Create a session

Create a session

PARAMETERS

Header Parameters

- Content-Type** string **Required**
Specifies the content type of the request body and should be set to application/json.
- X-Api-Version** integer **[2 .. 2] Required**
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.

REQUEST BODY

User credential.

- 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.

POST /rest/login-sessions/

REQUEST SAMPLES

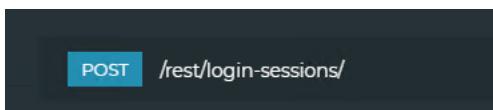
```
{ "authLoginDomain": "local", "password": "secret", "userName": "sarah" }
```

RESPONSE SAMPLES

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

```
{ "token": "NTAxMjQzNDc2NzIyR2yWViDr0E", "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 [2 .. 2] Required 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 Required 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 username to use for authentication
- **password**: set to the password for that username
- **authLoginDomain**: set the login domain of the user

REQUEST BODY

User credential.

userName	string Required User name.
password	string Required 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.

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



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

```
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.1j.lab/rest/login-sessions</code>
Verb	<code>POST</code>
Header	<code>Content-type:application/json</code> <code>X-API-Version:2</code>
Payload	<pre>{ "userName": "xxxxx", "password": "xxxx", "authLoginDomain": "lj.lab" }</pre>

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 method is set to POST and 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 method is set to POST and 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

`GET` Add appliance

`GET` ② Returns a list of appliances

`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 ③

- start** integer `>= 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 `>= 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": [
    {
      "uri": "/rest/appliances?count=1&start=0"
    }
  ],
  "modified": "2017-10-13T07:26:51Z",
  "nextPageUri": null,
  "prevPageUri": null,
  "start": 0,
  "total": 1
}
```



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

Header Parameters (?)

auth	string Required
	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 [2..2] Required 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	<code>https://oneview-global-dashboard.1j.lab/rest/appliances</code>
Verb	<code>GET</code>
Header	<code>Auth : your sessionID</code> <code>X-API-Version:2</code>



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

Then when all set, press **Send**

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

```

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 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.lj.lab/rest/resource-alerts</code>
Verb	GET
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 cpqSm2CntlrSelfTestErrors 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-CH75160600, 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      "severity": "Critical",
30      "type": "server-hardware",
31      "tag": "AlertResourceC3",
32      "serviceEventSource": false,
33      "lifeCycle": false,
34      "clearedTime": null,
35      "created": "2019-02-20T17:03:01.185Z",
36      "category": "alerts",
37      "eTag": "2019-02-20T17:03:01.185Z",
38      "activityURI": null,
39      "description": "Remote Insight/ Integrated Lights-Out self test error 8192.",
40      "id": "c0250109-8156-4e00-a117-8c77a1f3a364",
41      "originalURI": "/rest/alerts/43323",
42      "state": "Unknown",
43      "resourceUri": "/rest/appliances/85f94692-facc-460f-86fd-2d3fb91d9f30",
44      "applianceName": "Synergy Composer",
45      "applianceLocation": "192.168.1.110",
46      "associatedResourceName": "Frame1-CH75160600, bay 7",
47      "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 to the HPE OneView API, we use the same techniques, the same API patterns and like in OneView, the help in the documentation is very 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 run both or to jump to the scripting language of your choice.



PowerShell

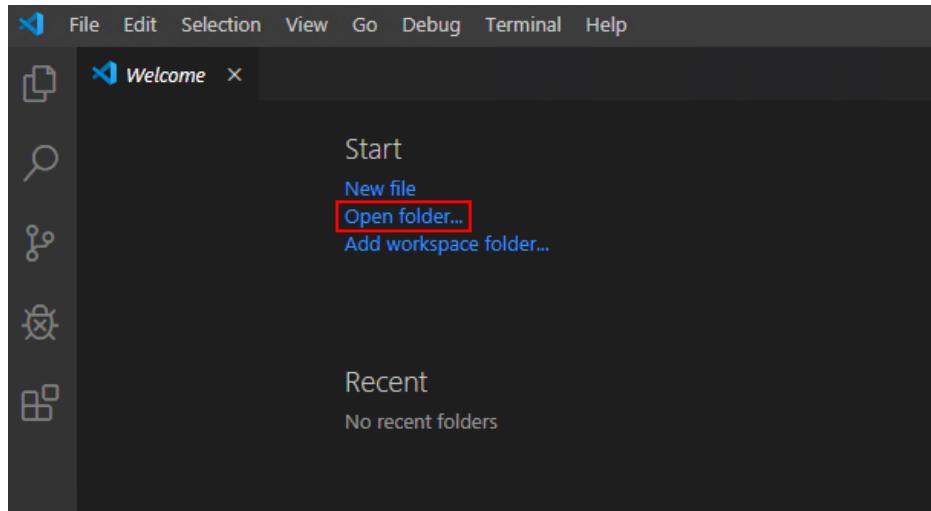
A PowerShell library for HPE OneView is available from HPE, it provides many functions to manage the appliance. We are not going to use this library in this lab as we want you to understand the way a REST call can be realized in PowerShell. Besides that, a library cannot provide all REST calls provided 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://hewlett-packard.github.io/POSH-HPOneView/>

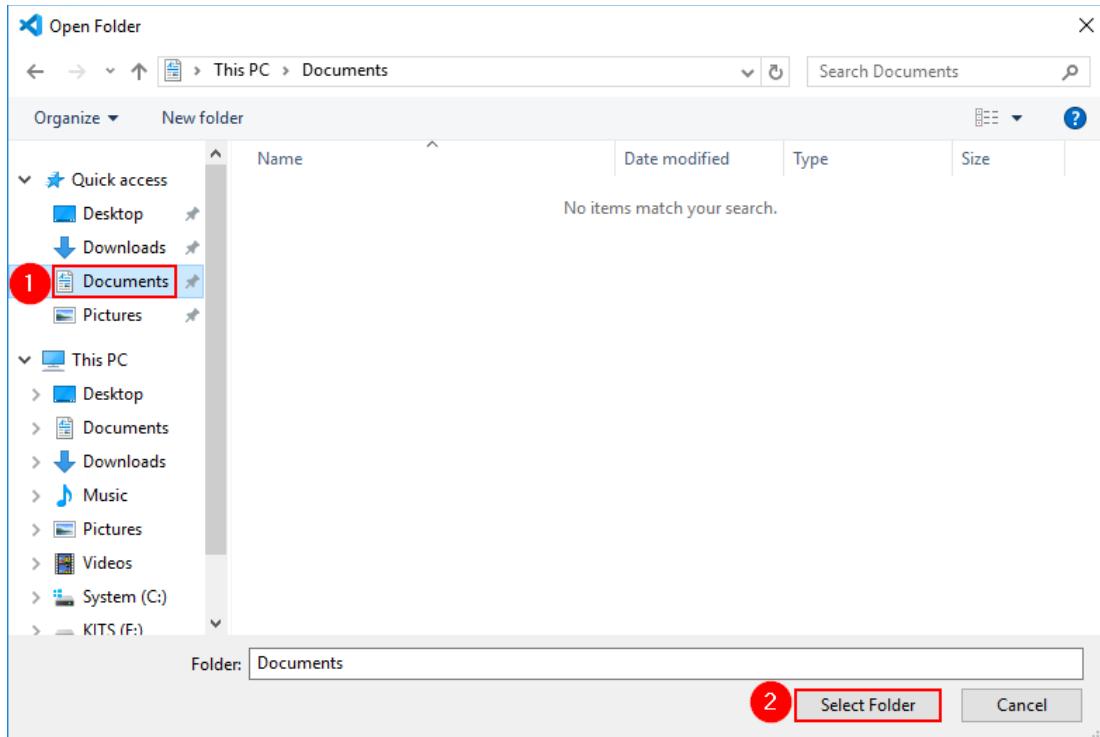
Start Visual Studio Code using the shortcut on the desktop:



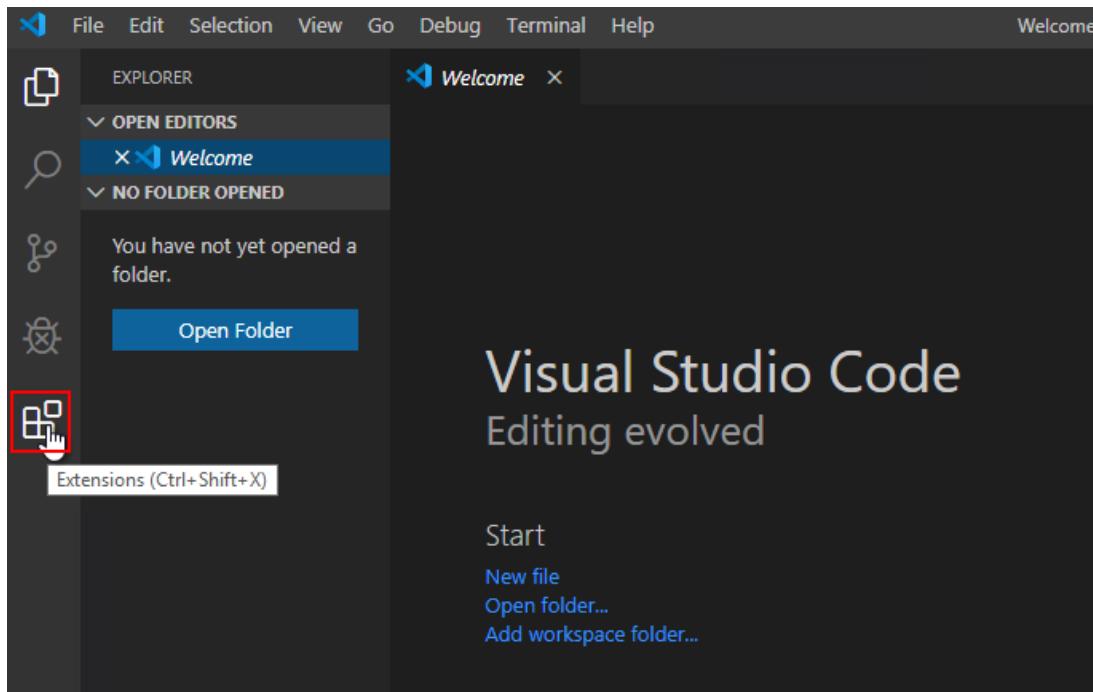
Click on **Open Folder...**



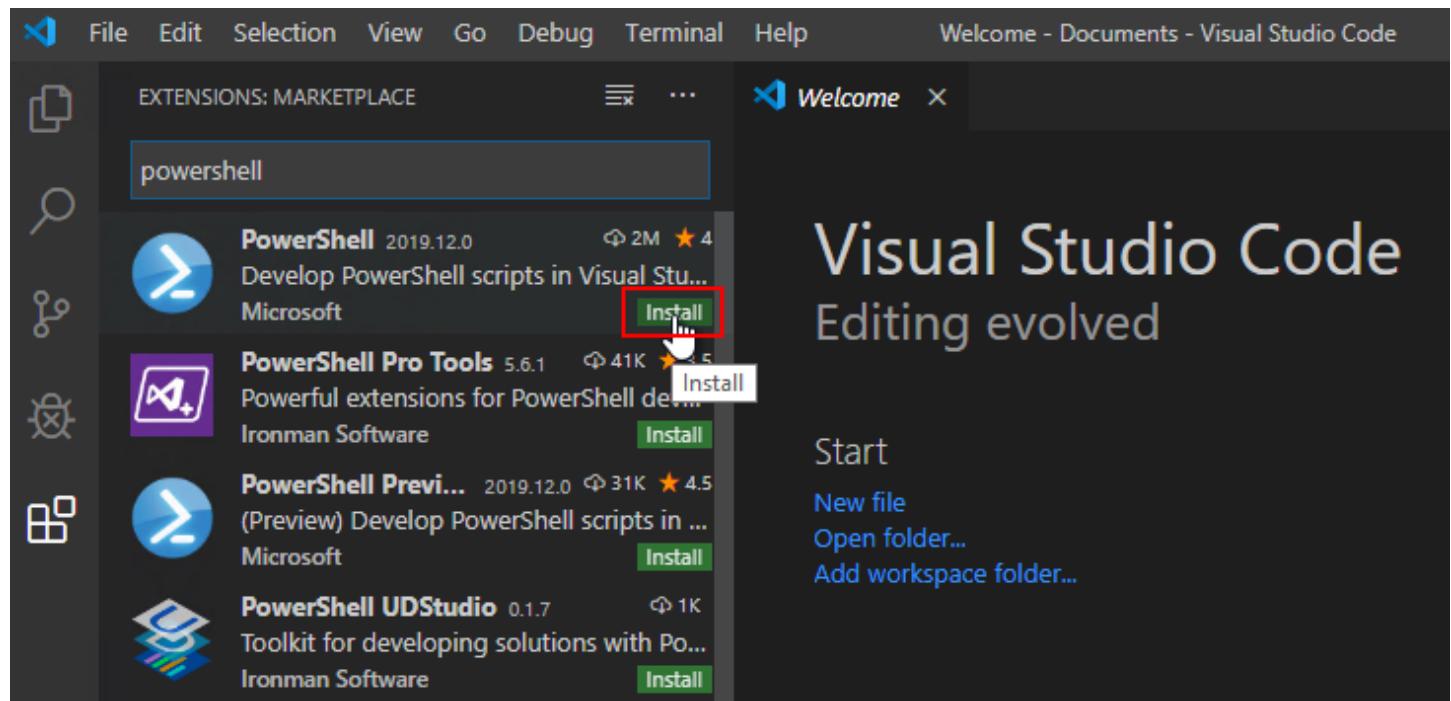
And select **Documents** and click **Select Folder**



Then click on **Extensions** on the Activity bar



If not already installed, install the *PowerShell* extension usually listed at the top of the list. You can enter **PowerShell** in the search tool to find the extension. Click on the **Install** button:



Visual Studio Code Editing evolved

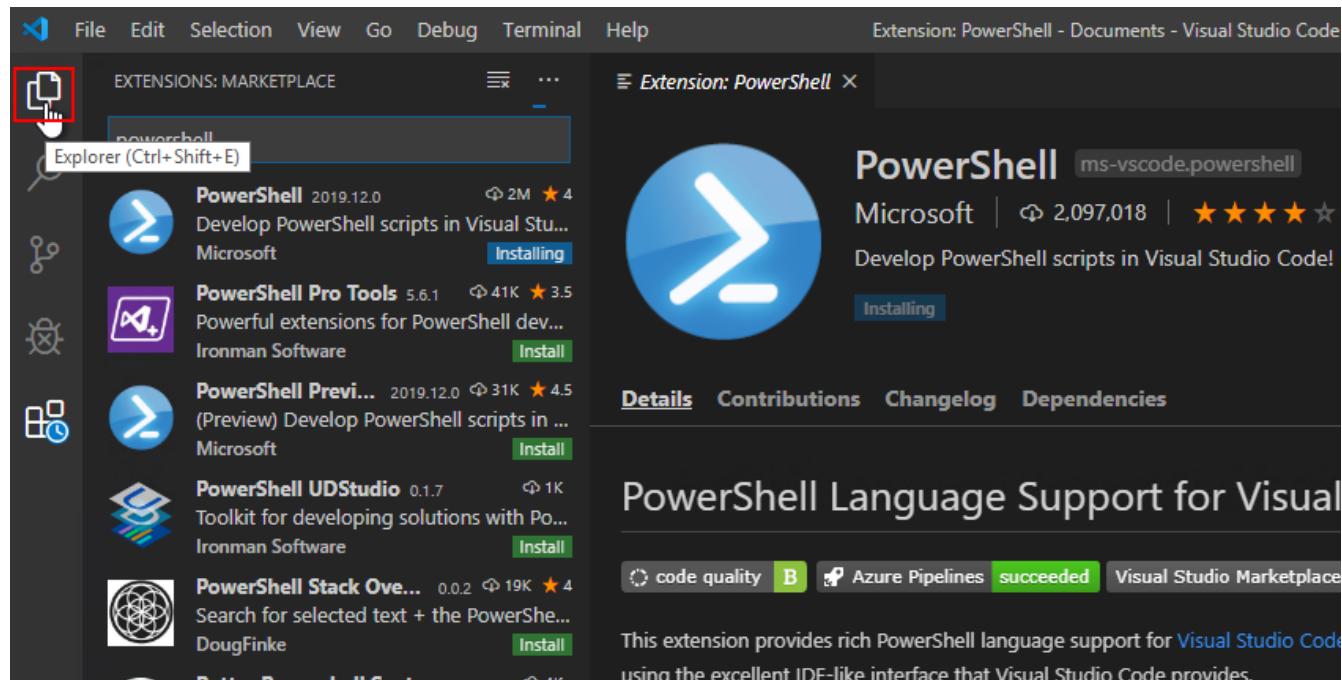
Start

New file

Open folder...

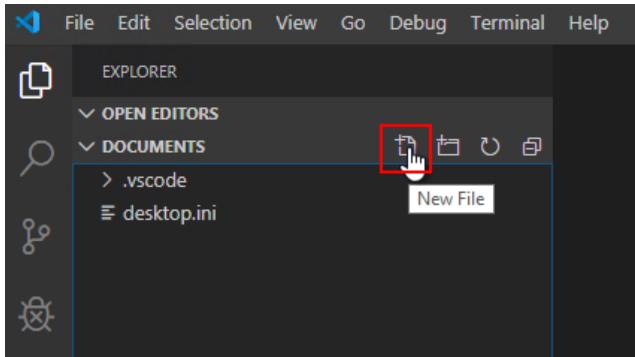
Add workspace folder...

While the extension is getting installed, click on **Explorer** on the Activity Bar

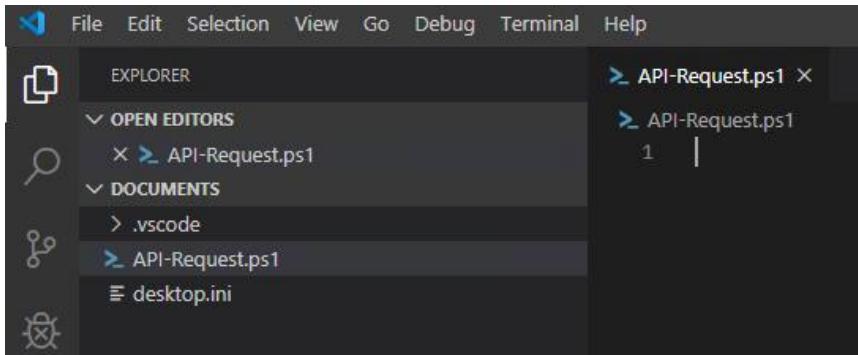




Then click on **New File**



Enter a name like **API-Request.ps1** then press **ENTER**



You can then enter the follow-on lines to build your PowerShell script.
The script we are going to use starts as follows:

```
#Defining the Global Dashboard credentials
$username = "team1"
$password = "Passwordteam1"
$globaldashboard = "oneview-global-dashboard.lj.lab"

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

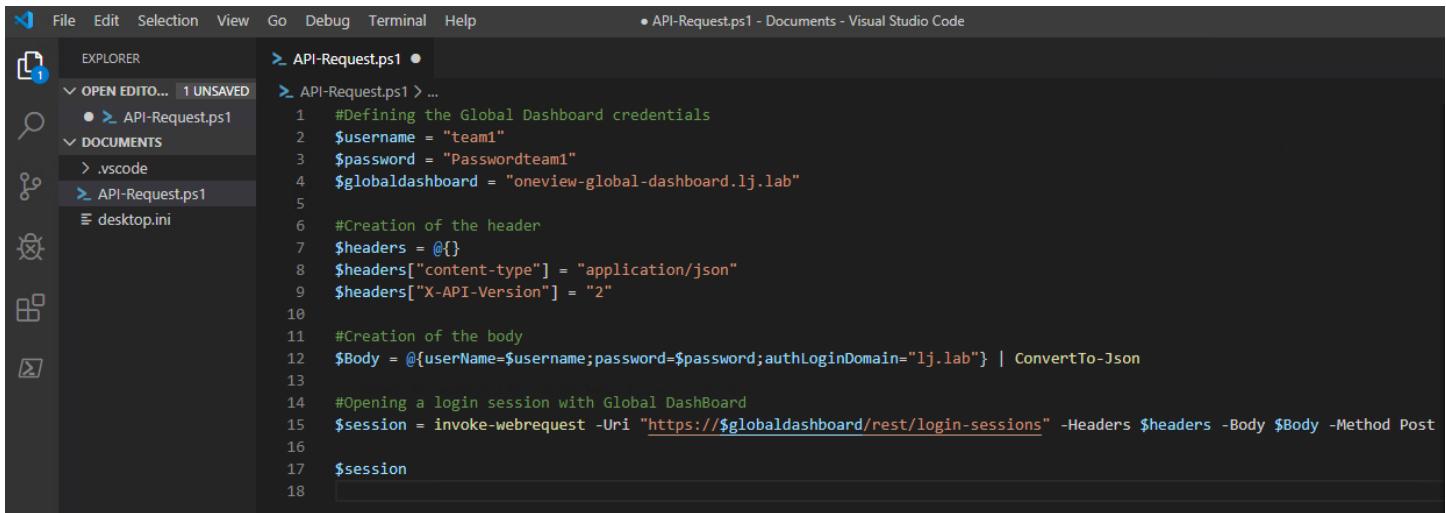
#Creation of the body
$Body = @{userName=$username;password=$password;authLoginDomain="lj.lab"} | ConvertTo-Json

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

$session
```



Just copy/paste these lines in VS Code:



```
#Defining the Global Dashboard credentials
$username = "team1"
$password = "Passwordteam1"
$globaldashboard = "oneview-global-dashboard.lj.lab"

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

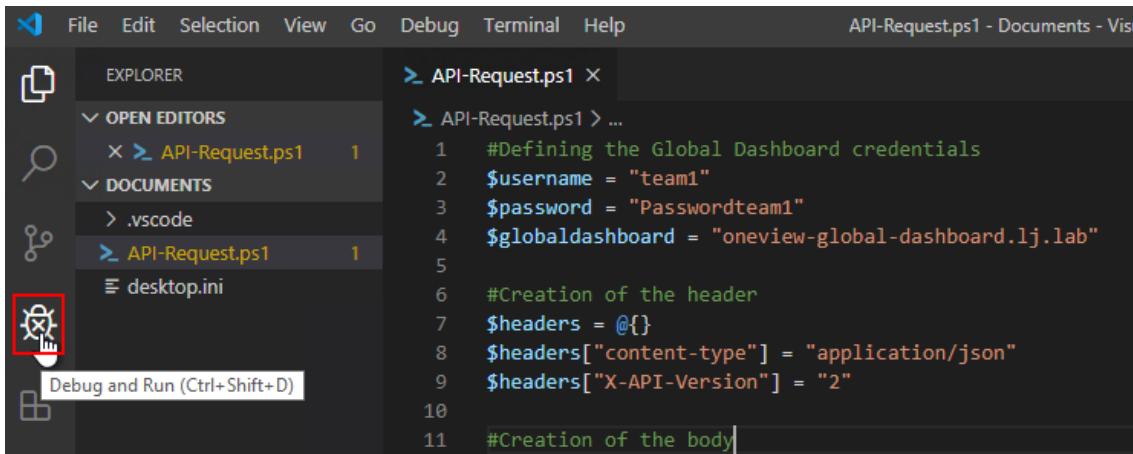
#Creation of the body
$Body = @{username=$username;password=$password;authLoginDomain="lj.lab"} | ConvertTo-Json

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

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, you can press **CTRL + S** to save the file.

Now let's run the script, click on **Debug and Run** on the Activity Bar



```
#Defining the Global Dashboard credentials
$username = "team1"
$password = "Passwordteam1"
$globaldashboard = "oneview-global-dashboard.lj.lab"

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

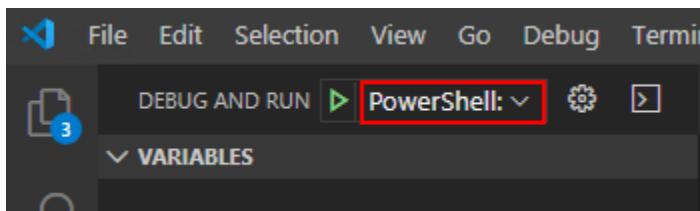
#Creation of the body
```

One of the key features of Visual Studio Code is its great debugging support. VS Code's built-in debugger helps accelerate edit, compile and debug script.

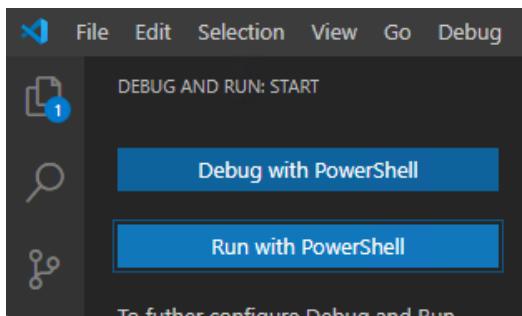


Visual Studio Code's built-in debugger needs some quick setup to run and debug PowerShell scripts. Please follow the procedure that applies to your environment:

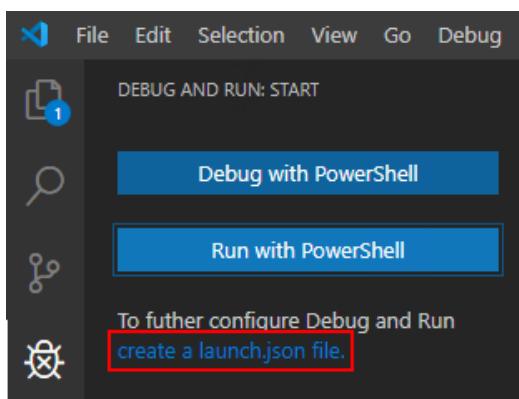
1. If PowerShell is displayed in the Debug and Run dialog, your environment is all set for PowerShell scripting, you can skip the setup section and step over points 2. and 3.:



2. If the following dialog is displayed in the Debug and Run pane:

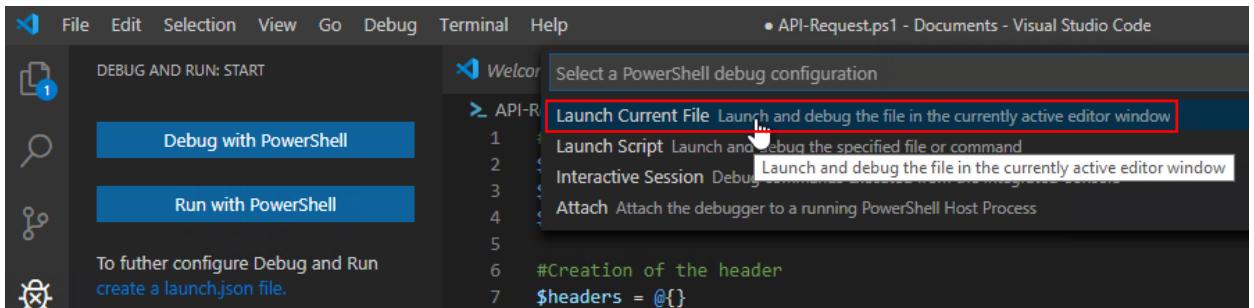


- a. Click on **Create a launch.json file** to configure Debug and Run





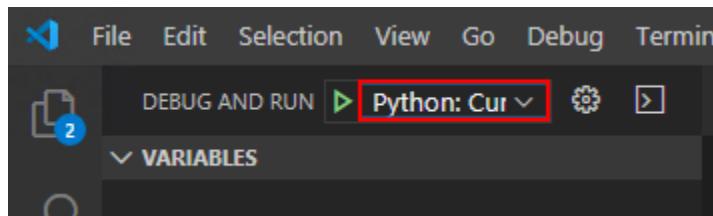
- b. Select the first option **Launch Current File**:



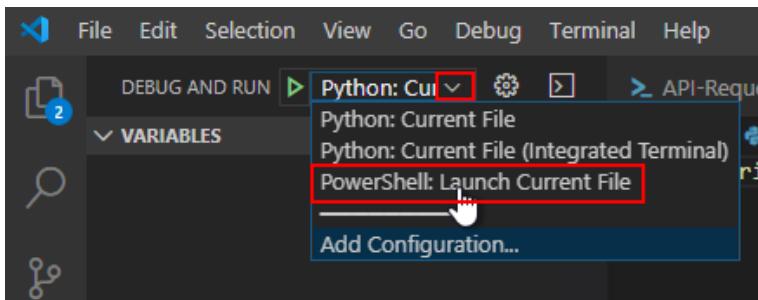
- c. Press **CTRL + S** to save the configuration file

- d. Press **CTRL + F4** to close the file

3. If Python is displayed in the Debug and Run dialog:

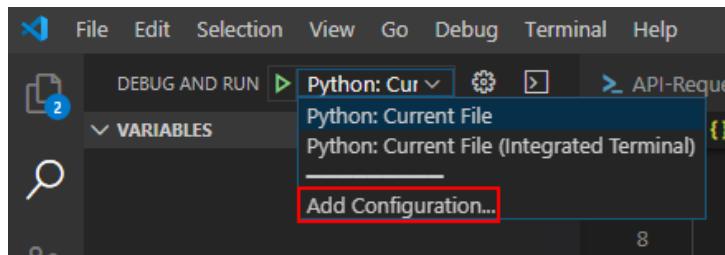


- a. Select **PowerShell: Launch Current File** from the Debug and Run drop-down menu and you are done.

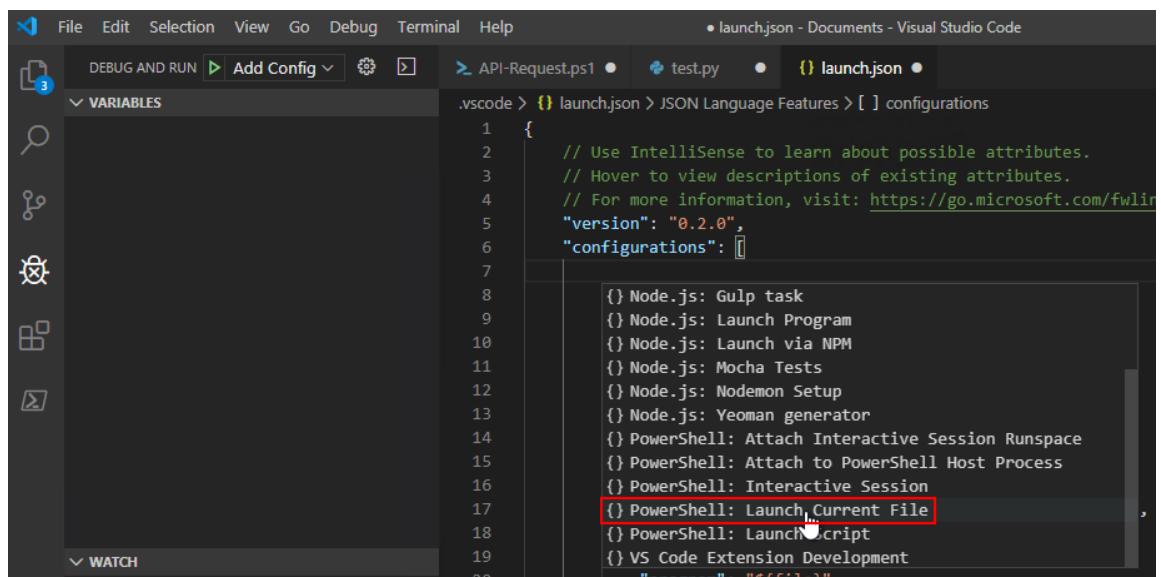




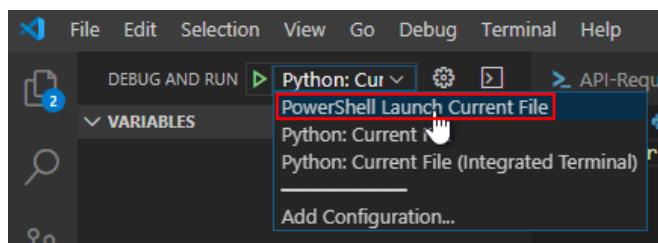
- i. If there is no PowerShell configuration, you need to select **Add configuration** from the drop-down menu:



- ii. Then select **PowerShell: Launch Current File**



- iii. Press **CTRL + S** to save the configuration file
- iv. Press **CTRL + F4** to close the file
- v. Then select **PowerShell: Launch Current File** from the Debug and Run drop-down menu



The Visual Studio Code debug and run setup is now complete, we can start debugging our script.



Using your mouse, drop a Breakpoint on the last line. Breakpoints are normally shown as red filled circles, can be toggled by clicking on the editor margin:

```
#Defining the Global Dashboard credentials
$username = "team1"
$password = "Passwordteam1"
$globaldashboard = "oneview-global-dashboard.lj.lab"

#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="GlobalDashboard";}

#Opening a login session with Global DashBoard
$session = invoke-webrequest -Uri "https://$globaldashboard/rest/v2/session" -Method Post -Body $Body -ContentType "application/json" -Headers $headers
```

Then click on **Start Debugging** to run the script

```
#Defining the Global Dashboard credentials
$username = "team1"
$password = "Passwordteam1"
$globaldashboard = "oneview-global-dashboard.lj.lab"

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



On the left-hand pane, the variables content is displayed, `$session` is the variable we are using to store the result of `invoke-webrequest`

The screenshot shows the Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Debug, Terminal, and Help. The title bar says "API-Request.ps1 - Documents - Visual Studio Code". The left sidebar has icons for file, folder, search, and variables. The "VARIABLES" section is expanded, showing various PowerShell variables like \$MaximumHistoryCount, \$NestedPromptLevel, \$OutputEncoding, \$PSDefaultParameterValues, \$PSEmailServer, \$PSSessionApplicationName, \$PSSessionConfigurationName, \$PSSessionOption, \$PWD, and \$session. The \$session variable is highlighted with a blue selection bar. The main editor area contains a PowerShell script named "API-Request.ps1". Lines 1 through 18 of the script are visible, starting with "#Defining the Global Dashboard credentials" and ending with "\$session".

```
#Defining the Global Dashboard credentials
$username = "team1"
$password = "Passwordteam1"
$globaldashboard = "oneview-global-dashboard.lj.lab"

#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="GlobalDashboard";}

#Opening a login session with Global DashBoard
$session = invoke-webrequest -Uri "https://$globaldashboard/api/session" -Method Post -Body $Body -ContentType "application/json" -Headers $headers -UseBasicParsing
```

You can discover the content of `$session` by using the expand icon:

This screenshot shows the same Visual Studio Code environment as above, but the \$session variable in the Variables sidebar is now expanded, indicated by a circled blue arrow icon. The expanded view shows the detailed structure of the \$session object, including its Content, ParsedHtml, Forms, InputFields, Links, Images, Scripts, AllElements, StatusCode, StatusDescription, RawContentStream, RawContentLength, RawContent, and BaseResponse properties.



A successful API request response code **200** should be displayed:

```

    < $session: {"user": {"user_name": "team1", "user_rol...
    | Content: {"user": {"user_name": "team1", "user_rol...
    > ParsedHtml: [HTMLDocumentClass]
    > Forms: [FormObjectCollection: 0]
    > InputFields: [WebCmdletElementCollection: 0]
    > Links: [WebCmdletElementCollection: 0]
    > Images: [WebCmdletElementCollection: 0]
    > Scripts: [WebCmdletElementCollection: 0]
    > AllElements: [WebCmdletElementCollection: 4]
    | StatusCode: 200
    | StatusDescription: "OK"
    > RawContentStream: [WebResponseContentMemoryStr...
    | RawContentLength: 215
    > RawContent: "HTTP/1.1 200 OK\r\ncontent-langua...

```

If you hover your mouse over the `content` description you will get the following popup box:

```

> $PSSessionOption: [PSSessionOption] 1 #Defining the Global Dashboard credentials
> $PWD: C:\Users\team2\Documents 2 $username = "team1"
> $session: {"user": {"user_name": "team1", "user_rol... 3 $password = "Passwordteam1"
    | Content: {"user": {"user_name": "team1", "user_rol... 4 $globaldashboard = "oneview-global-dashboard.lj.lab"
    > ParsedHtml: [HTMLDocumentClass] 5
    > Forms: [FormObjectCollection] 6 #Creation of the header
    > InputFields: [WebCmdletElementCollection] 7
    > Links: [WebCmdletElementCollection: 0] 8
    > RawContent: "HTTP/1.1 200 OK\r\ncontent-langua...

```

This is the response received from the API

```
{"user": {"user_name": "team1", "user_roles": ["Infrastructure administrator"], "domain": "lj.lab"}, "token": "Nzc1Mjc5NTk0NDAzxfT7RivXF_i4AT6YJ_ZRCw7iQmA6fGdh", "sessionId": "Nzc1Mjc5NTk0NDAzxfT7RivXF_i4AT6YJ_ZRCw7iQmA6fGdh"}"
```

Notice that the content of `content` is in JSON format. So, to access to the `sessionId`, we need to convert the response into a PowerShell object, easier to manipulate.

Click on **Continue** or **F5** to continue the script execution.

```

File Edit Selection View Go Debug Terminal Help
API-Request.ps1 - Documents - Visual Studio Code
DEBUG AND RUN PowerShell Launch Current Variables API-Request.ps1 ...
$PSSessionOption: [PSSessionOption]
$PWD: C:\Users\team2\Documents
1 #Defining the Global Dashboard credentials
2 $username = "team1"
3 $password = "Passwordteam1"
4 $globaldashboard = "oneview-global-dashboard.lj.lab"
5
6 #Creation of the header
7
8
9 $headers["X-API-Version"] = "2"
10

```



The last step of the script executes `$session`. This is another way to get the result of our API request displayed in the console:

A screenshot of a PowerShell terminal window. The terminal tab is labeled "TERMINAL". The command `$session` was run, and its output is displayed. The output shows various properties of the session object, such as StatusCode (200), StatusDescription (OK), Content (a JSON object containing user information like username, roles, domain, token, and session ID), RawContent (HTTP response headers), and Headers (content-language, Vary, Accept-Encoding, Keep-Alive, Connection). Other properties shown include Forms, Images, InputFields, Links, ParsedHtml, and RawContentLength.

```
11 #Creation of the body
12 $Body = @{userName=$username;password=$password;authLoginDomain="lj.lab"} | ConvertTo-Json
13
14 #Opening a login session with Global Dashboard

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
2: PowerShell Integrated Terminal +  [-]

[DBG]: PS C:\Users\team2\Documents>

StatusCode      : 200
StatusDescription : OK
Content          : {"user":{"userName":"team1","user_roles":["Infrastructure administrator"],"domain":"lj.lab"},"token":"Nzc1Mjc5NTk0NDAzxfT7RivXF_i4AT6YJ_ZRCw7iQmA6fGdh","sessionID":"Nzc1Mjc5NTk0NDAzxfT7RivXF_i4AT6YJ_Z..."}
RawContent       : HTTP/1.1 200 OK
                  content-language: en-us
                  Vary: Accept-Encoding
                  Keep-Alive: timeout=15, max=100
                  Connection: Keep-Alive
                  Content-Length: 215
                  Cache-Control: no-cache, private
                  Content-Type: applicati...
Forms            : {}
Headers          : {[content-language, en-us], [Vary, Accept-Encoding], [Keep-Alive, timeout=15, max=100], [Connection, Keep-Alive]...}
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 in the console:

```
$session.content
```

A screenshot of a PowerShell terminal window. The command `$session.content` was run, and its output is displayed. The output is a JSON object representing the session content, which includes the user object, token, and session ID.

```
PS C:\Users\team2\Documents> $session.content
{"user":{"userName":"team1","user_roles":["Infrastructure administrator"],"domain":"lj.lab"},"token":"Nzc1Mjc5NTk0NDAzxfT7RivXF_i4AT6YJ_ZRCw7iQmA6fGdh","sessionID":"Nzc1Mjc5NTk0NDAzxfT7RivXF_i4AT6YJ_ZRCw7iQmA6fGdh"}
PS C:\Users\team2\Documents>
```



So, the content of `content` is in JSON format. With PowerShell, we can easily convert JSON data into something easier to manipulate by using `convertfrom-json` cmdlet, enter:

```
$session.Content | ConvertFrom-Json
```

```
PS C:\Users\team2\Documents> $session.content | ConvertFrom-Json
user                                            token                                            sessionID
---                                            ----                                            -----
@{userName=team1; user_roles=System.Object[]; domain=lj.lab} Nzc1Mjc5NTk0NDAzxfT7RivXF_i4AT6YJ_ZRCw7iQmA6fGdh Nzc1Mjc5NTk0NDAzxfT7RivXF_i4AT6...
```

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\team2\Documents> ($session.Content | ConvertFrom-Json).token
Nzc1Mjc5NTk0NDAzxfT7RivXF_i4AT6YJ_ZRCw7iQmA6fGdh
```

Then we can easily capture this `sessionID` token in a variable `$key` for all API call methods we will do later on.

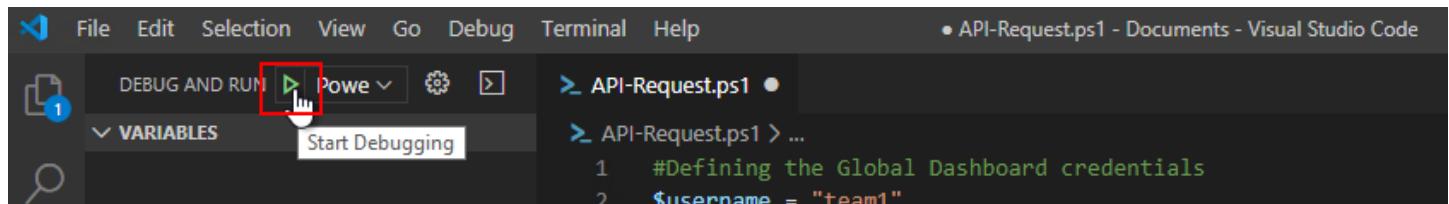
So, replace `$session` at the bottom of the script with the following lines:

```
#Capturing the OneView Global DashBoard Session ID and adding it to the header
$key = ($session.Content | ConvertFrom-Json).token
```

```
14 #Opening a login session with Global Dashboard
15 $session = invoke-webrequest -Uri "https://$globaldashboard/rest/login-sessions" -Headers $headers -Body $Body -Method Post
16
17 #Capturing the OneView Global DashBoard Session ID and adding it to the header
18 $key = ($session.Content | ConvertFrom-Json).token
19
20
```

Then you can remove the breakpoint by directly clicking on the breakpoint or by using the menu **Debug / Remove all Breakpoints**.

Then press **Start Debugging** to run the script again with the two new lines





Now type \$key in the console and press Enter, it should display the Global DashBoard authentication ID.

```
PS C:\Users\team2\Documents> $key
Mjg5MjUwNjcyNTM0nERXhv60aem9pnHhvW42HrWb4jV2MXr9
```

Then we can use this \$key to modify our header. If you enter \$headers, you get two items:

```
PS C:\Users\team2\Documents> $headers
Name          Value
----          -----
X-API-Version 2
content-type   application/json
```

Let's add the auth item to the header with the \$key value using:

```
$headers["auth"] = $key
```

If you re-enter \$headers, we see now that auth has been added with our session ID.

```
PS C:\Users\team2\Documents> $headers["auth"] = $key
PS C:\Users\team2\Documents> $headers
Name          Value
----          -----
auth          Mjg5MjUwNjcyNTM0nERXhv60aem9pnHhvW42HrWb4jV2MXr9
content-type   application/json
X-API-Version 2
```

Add \$headers["auth"] = \$key at the bottom of the script:

```
17 #Capturing the OneView Global Dashboard Session ID and adding it to the header
18 $key = ($session.content | ConvertFrom-Json).token
19
20 $headers["auth"] = $key
```

Now every Global Dashboard API call method we do will use this header with the auth token for the authentication.

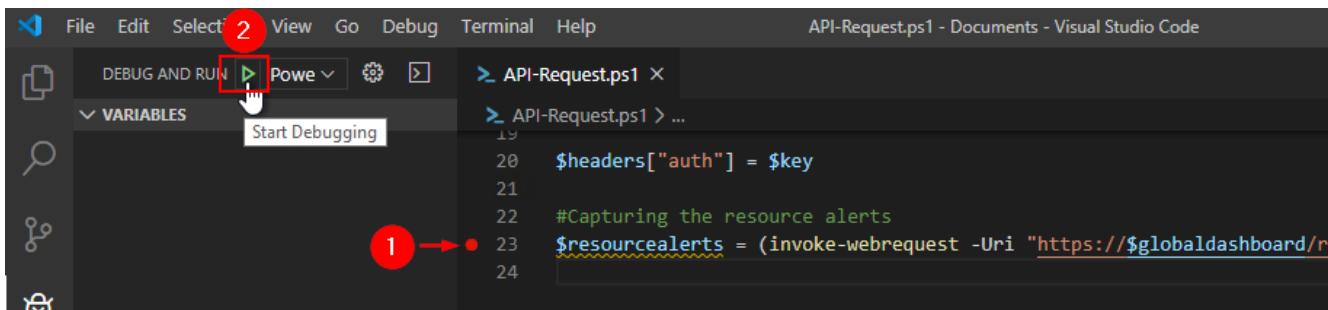
As an example, we can use the GET /rest/resource-alerts method to retrieve the critical alerts available in Global Dashboard, we will use once again invoke-webrequest, enter now:

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

Note: make sure you type or copy/paste this line as a single line



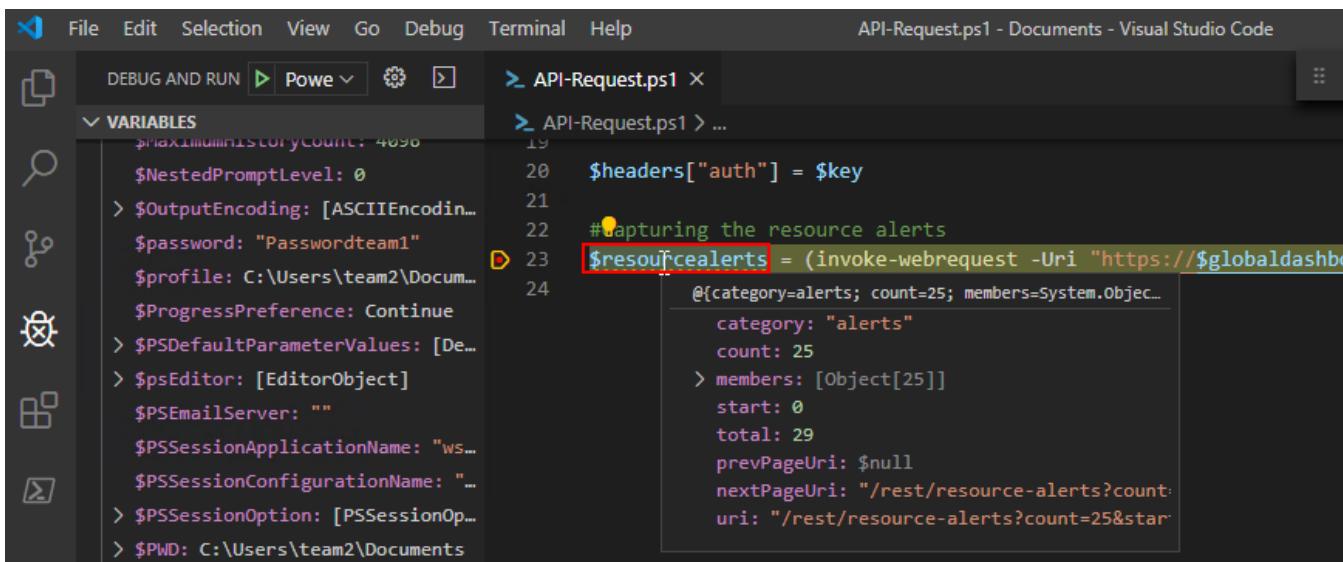
Add a breakpoint on the last line and click on **Start Debugging**



```
File Edit Selection View Go Debug Terminal Help API-Request.ps1 - Documents - Visual Studio Code
DEBUG AND RUN Power ▾ Variables API-Request.ps1 > ...
VARIABLES Start Debugging
19
20 $headers["auth"] = $key
21
22 #Capturing the resource alerts
23 $resourcealerts = (invoke-webrequest -Uri "https://$globaldashboard/r
24
```

The result of the webrequest is saved in `$resourcealerts`

You can see the content of `$resourcealerts` by hovering your mouse over `$resourcealerts`



```
File Edit Selection View Go Debug Terminal Help API-Request.ps1 - Documents - Visual Studio Code
DEBUG AND RUN Power ▾ Variables API-Request.ps1 > ...
VARIABLES $maxHistoryCount: 4096
$NestedPromptLevel: 0
> $OutputEncoding: [ASCIIEncoding]
$password: "Passwordteam1"
$profile: C:\Users\team2\Documents
$ProgressPreference: Continue
> $PSDefaultParameterValues: [De...
> $psEditor: [EditorObject]
$PSEmailServer: ""
$PSSessionApplicationName: "ws...
$PSSessionConfigurationName: ...
> $PSSessionOption: [PSSessionOp...
> $PWD: C:\Users\team2\Documents
$resourcealerts = (invoke-webrequest -Uri "https://$globaldashboard/r
@{category=alerts; count=25; members=System.Object[]}
category: "alerts"
count: 25
> members: [Object[25]]
start: 0
total: 29
prevPageUri: $null
nextPageUri: "/rest/resource-alerts?count=25&start=25"
uri: "/rest/resource-alerts?count=25&start=25"
```

We see that the alerts are grouped into a list, sometimes referred as a collection. This list is called *members*.



To better understand its content, expand *members*

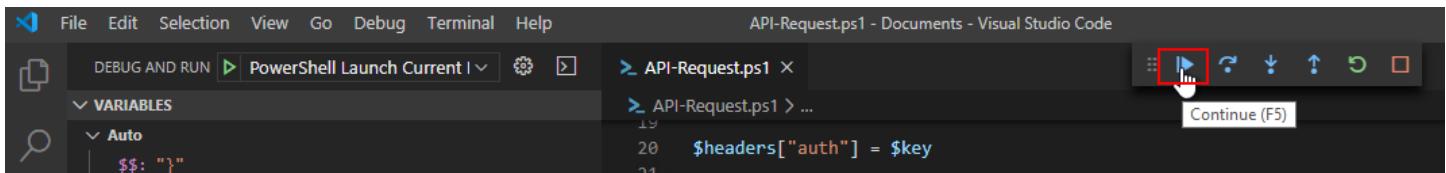
```
22 #Capturing the resource alerts
23 $resourcealerts = (invoke-webrequest -Uri "https://$globaldashboard/
24                                     @(category=alerts; count=25; members=System.Object[])
    category: "alerts"
    count: 25
    members: [Object[25]]
    ↘ [0]: @{type=AlertResourceV3; uri=/rest/r
    > [1]: @{type=AlertResourceV3; uri=/rest/r
    > [2]: @{type=AlertResourceV3; uri=/rest/r
    > [3]: @{type=AlertResourceV3; uri=/rest/r
    > [4]: @{type=AlertResourceV3; uri=/rest/r
    > [5]: @{type=AlertResourceV3; uri=/rest/r
    > [6]: @{type=AlertResourceV3; uri=/rest/r
    > [7]: @{type=AlertResourceV3; uri=/rest/r
    > [8]: @{type=AlertResourceV3; uri=/rest/r
    > [9]: @{type=AlertResourceV3; uri=/rest/r
    > [10]: @{type=AlertResourceV3; uri=/rest/
    > [11]: @{type=AlertResourceV3; uri=/rest/
    > [12]: @{type=AlertResourceV3; uri=/rest/
    > [13]: @{type=AlertResourceV3; uri=/rest/
    > [14]: @{type=AlertResourceV3; uri=/rest/
```

In this list, we have several objects (25), each representing an alert.

Then each object is represented by a paired list of “items”: “value”

```
22 #Capturing the resource alerts
23 $resourcealerts = (invoke-webrequest -Uri "https://$globaldashboard/res
24                                     @(category=alerts; count=25; members=System.Object[])
    category: "alerts"
    count: 25
    members: [Object[25]]
    ↘ [0]: @{type=AlertResourceV3; uri=/rest/r
    type: "AlertResourceV3"
    uri: "/rest/resource-alerts/d0c6de33-d0
    category: "alerts"
    eTag: "2020-01-03T17:31:46.264Z"
    created: "2020-01-03T17:31:46.264Z"
    modified: "2020-01-03T17:31:46.264Z"
    > associatedEventUris: [Object[1]]
    resourceId: "657e1907-8aea-4ba5-8b82-86
    assignedToUser: $null
    activityUri: $null
    > changelog: [Object[0]]
    clearedByUser: $null
    clearedTime: $null
    correctiveAction: "Verify that downlink"
```

Click now on **Continue**



The screenshot shows the Visual Studio Code interface with the following details:

- File menu: File, Edit, Selection, View, Go, Debug, Terminal, Help.
- Toolbar: DEBUG AND RUN (highlighted), PowerShell Launch Current, Run, Stop, Break, Continue (F5) (highlighted).
- Variables sidebar: VARIABLES section expanded, Auto variable expanded, value: \$\$: "{}".
- Code editor: API-Request.ps1 - Documents - Visual Studio Code. The code is as follows:

```
19
20     $headers["auth"] = $key
21
```



Another way to see the content information of `$resourcealerts` is to use `get-member` in the console, enter:

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

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL 2: PowerShell Integrated Terminal + □

Hit Line breakpoint on 'C:\Users\team2\Documents\API-Request.ps1:23'

[DBG]: PS C:\Users\team2\Documents> $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=25
members   NoteProperty Object[] members=System.Object[]
nextPageUri NoteProperty string nextPageUri=/rest/resource-alerts?count=25&start=25&query='severity%20EQ%20%22C...
prevPageUri NoteProperty object prevPageUri=null
start    NoteProperty int start=0
total    NoteProperty int total=29
uri      NoteProperty string uri=/rest/resource-alerts?count=25&start=0&query='severity%20EQ%20%22Critical%22'
```

`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 the `members` property.

To access the `members` property, it's easy, just enter:

```
$resourcealerts.members
```

```
[DBG]: PS C:\Users\team2\Documents> $resourcealerts.members

type          : AlertResourceV3
uri           : /rest/resource-alerts/d0c6de33-d008-40ae-95b7-1537cb2a9465
category      : alerts
eTag          : 2020-01-03T17:31:46.264Z
created       : 2020-01-03T17:31:46.264Z
modified      : 2020-01-03T17:31:46.264Z
associatedEventUris : {/rest/events/1915015}
resourceID    : 657e1907-8aea-4ba5-8b82-86582a3a3254
assignedToUser :
activityUri   :
changeLog     :
clearedByUser :
clearedTime   :
correctiveAction : Verify that downlink port is enabled and link status is linked. Verify that the interconnect link topology is valid, the interconnect is in a configured state and there are no network connectivity issues between OneView and the interconnect. If the problem persists, contact your authorized support representative and provide them with a support dump.
description   : An error has occurred on connection 8. Interconnect {"name":"Frame2,
```



The list can be very long and not easy to read so let's work on methods to get a better presentation.

We can retrieve the members of this object using this time:

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

```
[DBG]: PS C:\Users\team2\Documents> $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 object activityUri=null
alertState    NoteProperty string alertState=Locked
alertTypeID   NoteProperty string alertTypeID=profilemgr.Connections.CONNECTION_SCMB_ERROR
applianceLocation NoteProperty string applianceLocation=192.168.1.110
applianceModel  NoteProperty string applianceModel=Synergy Composer
applianceName   NoteProperty string applianceName=composer
applianceVersion NoteProperty string applianceVersion=5.00.01-0410269
appluri       NoteProperty string appluri=/rest/appliances/ec4638ac-5cdb-4baa-b0ed-2fb90dd1664c
assignedToUser NoteProperty object assignedToUser=null
associatedEventUris NoteProperty Object[] associatedEventUris=System.Object[]
associatedResource NoteProperty System.Management.Automation.PSCustomObject associatedResource=@{res...
associatedResourceName NoteProperty string associatedResourceName=win-1
associatedResourceOriginalUri NoteProperty string associatedResourceOriginalUri=/rest/server-profiles/a42f4b31-...
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=Verify that downlink port is enabled and lin...
created        NoteProperty string created=2020-01-03T17:31:46.264Z
description    NoteProperty string description=An error has occurred on connection 8. Interconn...
eTag          NoteProperty string eTag=2020-01-03T17:31:46.264Z
healthCategory NoteProperty string healthCategory=Connection
id            NoteProperty string id=d0c6de33-d008-40ae-95b7-1537cb2a9465
lifeCycle      NoteProperty bool lifeCycle=False
modified       NoteProperty string modified=2020-01-03T17:31:46.264Z
originalUri   NoteProperty string originalUri=/rest/alerts/27183
physicalResourceType NoteProperty string physicalResourceType=server-profiles
resourceID    NoteProperty string resourceID=657e1907-8aea-4ba5-8b82-86582a3a3254
resourceUri   NoteProperty string resourceUri=/rest/server-profiles/a42f4b31-ca97-44b8-8b72-4c7...
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=High
uri           NoteProperty string uri=/rest/resource-alerts/d0c6de33-d008-40ae-95b7-1537cb2a9465
```

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 use the cmdlet *Format-List* to display only the `severity`, `description` and `correctiveAction` properties, enter in the console:

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

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

```
[DBG]: PS C:\Users\team2\Documents> $resourcealerts.members | Format-List -Property severity,description,correctiveAction

severity      : Critical
description   : An error has occurred on connection 8. Interconnect {"name":"Frame2, interconnect 6","uri":"/rest/interconnects/4b94ebc4-5bc3-447b-9b6b-66b3f9761bb7"} port 25 subport b is unlinked.
correctiveAction : Verify that downlink port is enabled and link status is linked. Verify that the interconnect link topology is valid, the interconnect is in a configured state and there are no network connectivity issues between OneView and the interconnect. If the problem persists, contact your authorized support representative and provide them with a support dump.

severity      : Critical
description   : An error has occurred on connection 6. Interconnect {"name":"Frame2, interconnect 6","uri":"/rest/interconnects/4b94ebc4-5bc3-447b-9b6b-66b3f9761bb7"} port 25 subport d is unlinked.
correctiveAction : Verify that downlink port is enabled and link status is linked. Verify that the interconnect link topology is valid, the interconnect is in a configured state and there are no network connectivity issues between OneView and the interconnect. If the problem persists, contact your authorized support representative and provide them with a support dump.

severity      : Critical
description   : Connection on downlink port 25, subport b has failed. The subport is unlinked.
correctiveAction : Verify that the downlink port is enabled and link status is linked. Verify that there are no critical interconnect link topology alerts for the interconnect {"name":"Frame2, interconnect 6","uri":"/rest/interconnects/4b94ebc4-5bc3-447b-9b6b-66b3f9761bb7"}. If a critical interconnect link topology alert exists, follow the resolution steps in the alert to restore the interconnect link topology. Verify that the interconnect is in a configured state and that there are no network connectivity problems between OneView and the interconnect. If the problem persists, contact your authorized support representative and provide a support dump.
```



So finally, add to your script the following lines to complete the script.:

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

Write-host "`nThe latest 5 alerts are: "

$resourcealerts.members | Select-Object -First 5 | Format-List -Property
severity,description,correctiveAction
```

The cmdlet *Select-Object* is used to only display the first 5 alerts.

Note: make sure you type or copy/paste this last line as a single line in VS Code

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

Once the three lines are added, you can remove all breakpoints you may have and re-run the script to see the result:

The screenshot shows the VS Code interface with the PowerShell Integrated terminal active. The terminal window displays the execution of a PowerShell script. The output shows the count of alerts (25), followed by the details of the top five alerts, including severity, description, and corrective actions. The terminal tab bar shows other tabs like PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL.

```
22 #Capturing the resource alerts
23 $resourcealerts = (invoke-webrequest -Uri "https://$globaldashboard/rest/resource-alerts" -Header
24
25
26 Write-host "The number of alerts is" $resourcealerts.count
27 Write-host "`nThe latest 5 alerts are: "
28 $resourcealerts.members | Select-Object -First 5 | Format-List -Property severity,description,cor

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: PowerShell Integrated + □

The number of alerts is 25
The latest 5 alerts are:

severity      : Critical
description   : An error has occurred on connection 8. Interconnect {"name":"Frame2, interconnect
                 6","uri":"/rest/interconnects/4b94ebc4-5bc3-447b-9b6b-66b3f9761bb7"} port 25 subport b is
                 unlinked.
correctiveAction : Verify that downlink port is enabled and link status is linked. Verify that the interconnect
                   link topology is valid, the interconnect is in a configured state and there are no network
                   connectivity issues between OneView and the interconnect. If the problem persists, contact
                   your authorized support representative and provide them with a support dump.

severity      : Critical
description   : An error has occurred on connection 6. Interconnect {"name":"Frame2, interconnect
                 6","uri":"/rest/interconnects/4b94ebc4-5bc3-447b-9b6b-66b3f9761bb7"} port 25 subport d is
                 unlinked.
correctiveAction : Verify that downlink port is enabled and link status is linked. Verify that the interconnect
                   link topology is valid, the interconnect is in a configured state and there are no network
                   connectivity issues between OneView and the interconnect. If the problem persists, contact
                   your authorized support representative and provide them with a support dump.

severity      : Critical
description   : Connection on downlink port 25, subport b has failed. The subport is unlinked.
correctiveAction : Verify that the downlink port is enabled and link status is linked. Verify that there are no
                   critical interconnect link topology alerts for the interconnect {"name":"Frame2, interconnect
                   6","uri":"/rest/interconnects/4b94ebc4-5bc3-447b-9b6b-66b3f9761bb7"}. If a critical
                   interconnect link topology alert exists, follow the resolution steps in the alert to restore
                   the interconnect link topology. Verify that the interconnect is in a configured state and that
                   there are no network connectivity problems between OneView and the interconnect. If the
                   problem persists, contact your authorized support representative and provide a support dump.
```

This concludes the PowerShell Lab.



Python

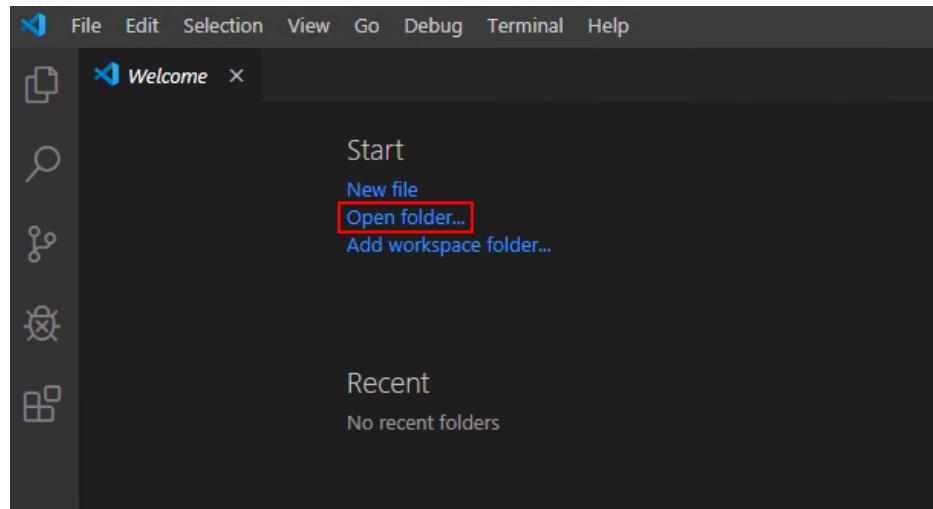
A Python library for HPE OneView is available from HPE, it provides basic functions to manage the appliance. We are not going to use this library in this lab as we want you to understand the way a REST call can be realized in Python. Besides that, a library cannot provide all REST calls provided by an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

Start Visual Studio Code using the shortcut on the desktop:

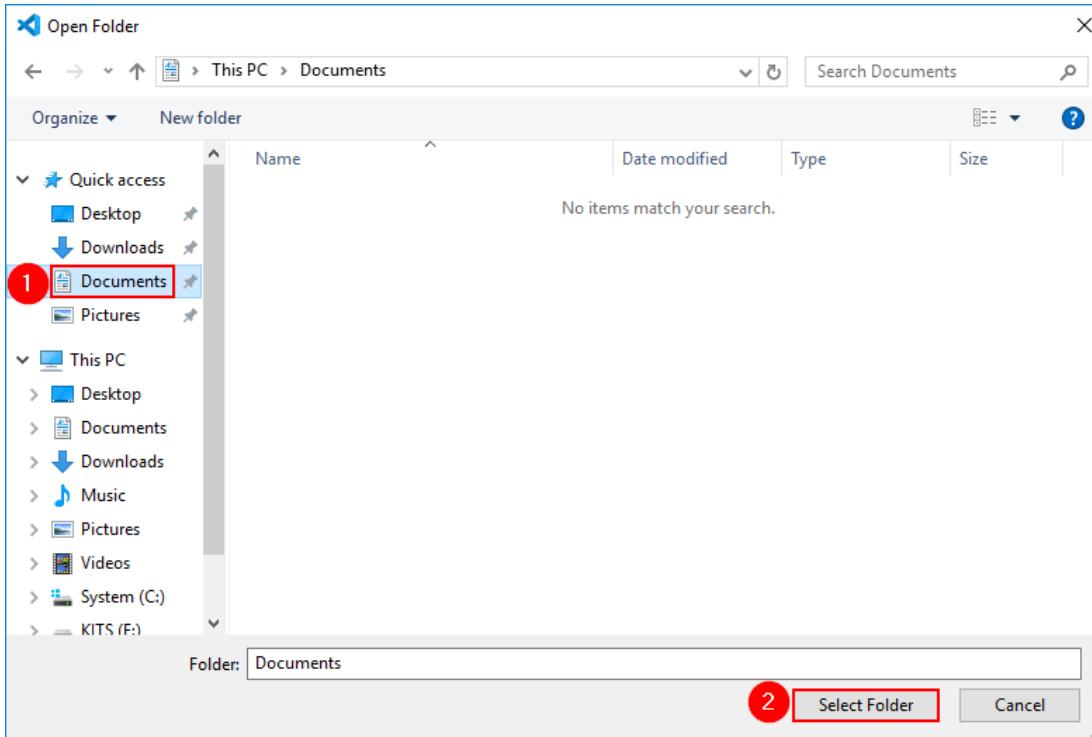


Click on **Open Folder...**

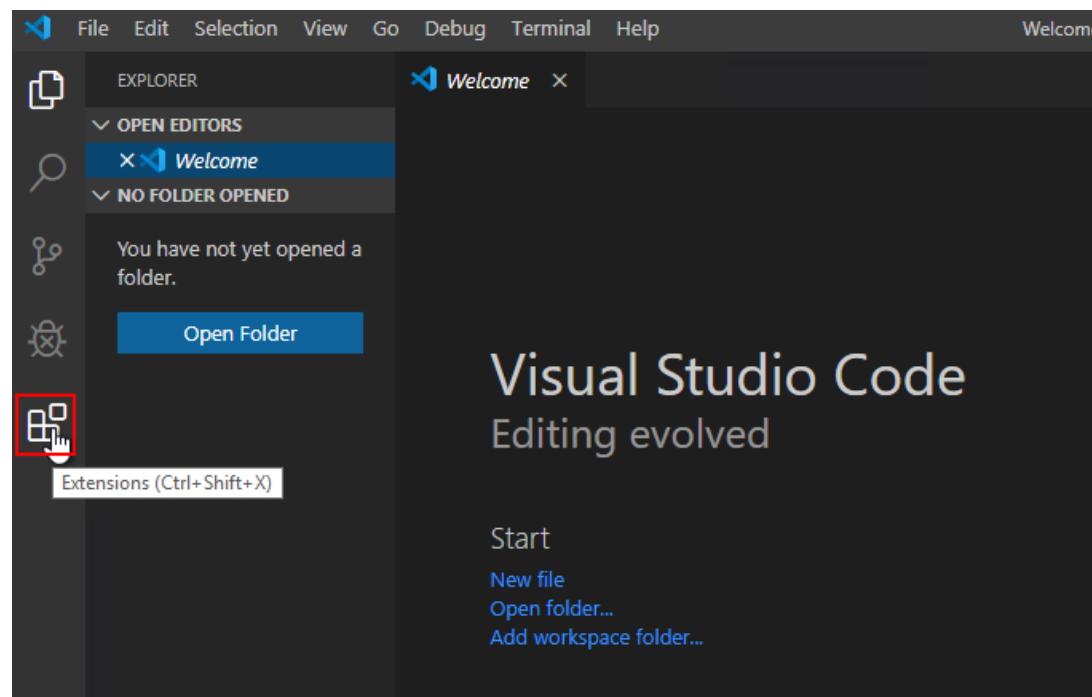




And select **Documents** and click **Select Folder**

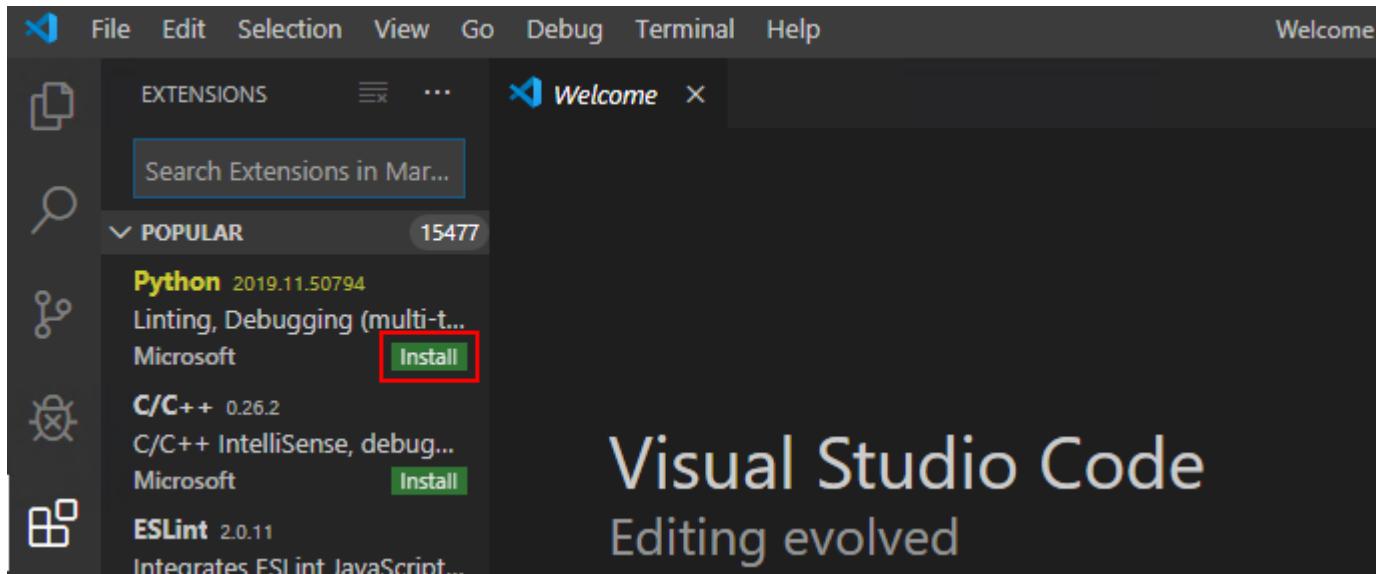


Then click on **Extensions** on the Activity bar





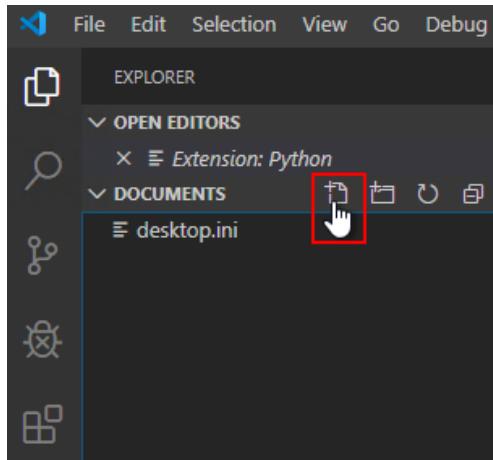
If not already installed, install the *Python* extension usually listed at the top of the list. You can enter **python** in the search tool to find the extension. Click on the **Install** button:



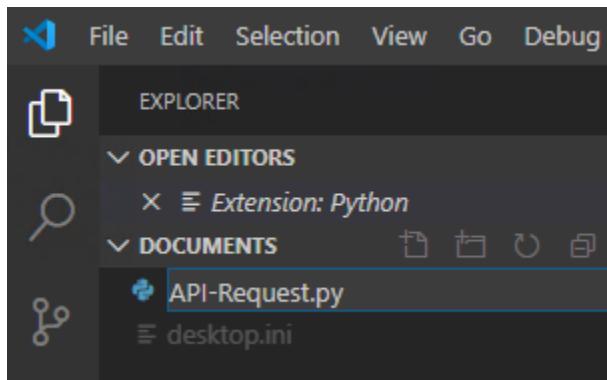
While the extension is getting installed, click on **Explorer** on the Activity Bar



Then click on **New File**

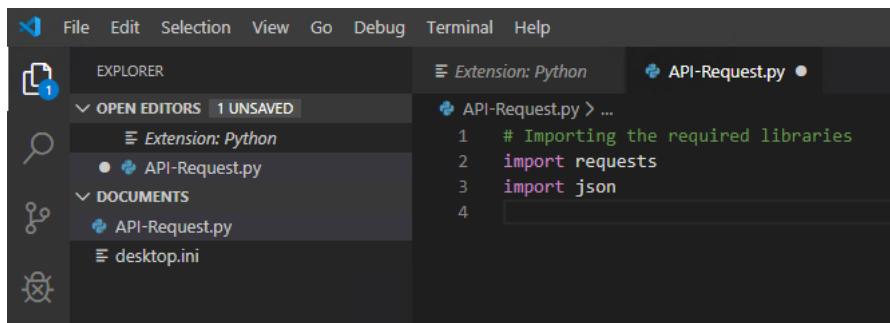


Enter a name like **API-Request.py** then press **ENTER**



You can then enter the follow-on lines to build your Python script.
The script we are going to use starts as follows, just copy/paste these lines in VS Code:

```
# Importing the required libraries
import requests
import json
```





The `requests` 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 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.

A second function `json` is also imported, we will use it to convert the Global DashBoard json response into a python dictionary, much easier to manipulate.

Next, we are defining our Global DashBoard credentials in variables

```
# Defining the Global Dashboard credentials
username = "<username>"
password = "<password>"
globaldashboard = "oneview-global-dashboard.lj.lab"
```

You need to update `<username>` and `<password>` with your provided credentials.

Next, enter the following lines to create the body and header content:

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

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

The script just follows the same steps we saw previously with Postman: the creation of the body for the login session (with `userName/password/authLoginDomain`) with the creation of the header (with `content-type` and `x-api-version`)

Next, we can finally create the API query using the Python `requests` function using the `POST` action with our body, header and URL then we display on the console the `response` content:

```
# Creating the web request with body and header content
response = requests.post(
    url="https://" + globaldashboard + "/rest/login-sessions",
    verify=False,
    json=body,
    headers=header
)

print(response)
```



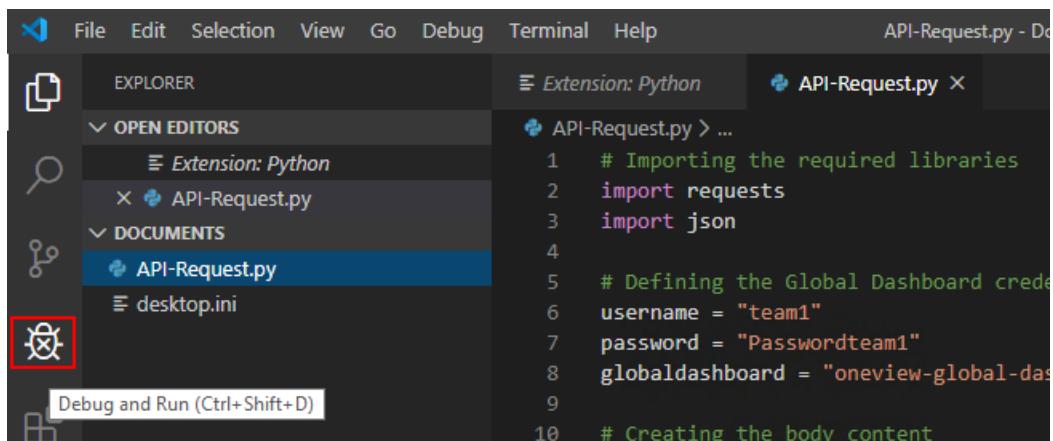
Your VS Code script should look like this:

```
Extension: Python API-Request.py ●

API-Request.py > ...
1 # Importing the required libraries
2 import requests
3 import json
4
5 # Defining the Global Dashboard credentials
6 username = "team1"
7 password = "Passwordteam1"
8 globaldashboard = "oneview-global-dashboard.lj.lab"
9
10 # Creating the body content
11 body = {
12     'userName': username,
13     'password': password,
14     'authLoginDomain': 'lj.lab'
15 }
16
17 # Creating the header content
18 header= {
19     'Content-Type': 'application/json',
20     'X-API-VERSION': '2'
21 }
22
23 # Creating the web request with body and header content
24 response = requests.post(
25     url="https://" + globaldashboard + "/rest/login-sessions",
26     verify=False,
27     json=body,
28     headers=header
29 )
30
31 print(response)
```

You can press **CTRL + S** to save the file.

Now let's run the script, click on **Debug and Run** on the Activity Bar

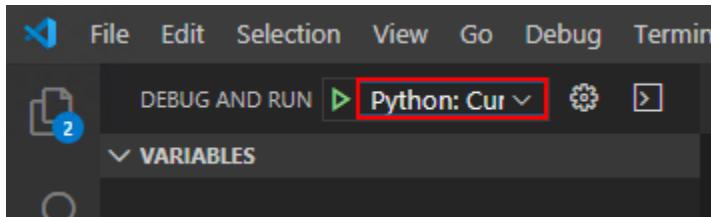




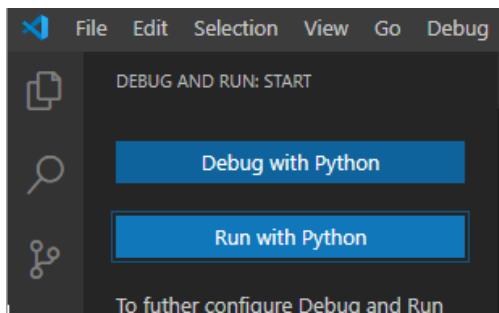
One of the key features of Visual Studio Code is its great debugging support. VS Code's built-in debugger helps accelerate edit, compile and debug script.

Visual Studio Code's built-in debugger needs some quick setup to run and debug Python scripts. Please follow the procedure that applies to your environment:

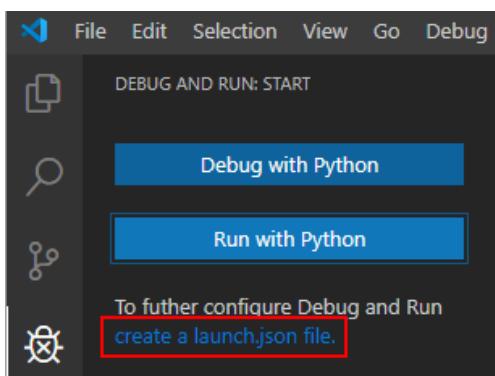
1. If Python is displayed in the Debug and Run dialog, your environment is all set for Python scripting, you can skip the setup section and step over points 2. and 3.:



2. If the following dialog is displayed in the Debug and Run pane:

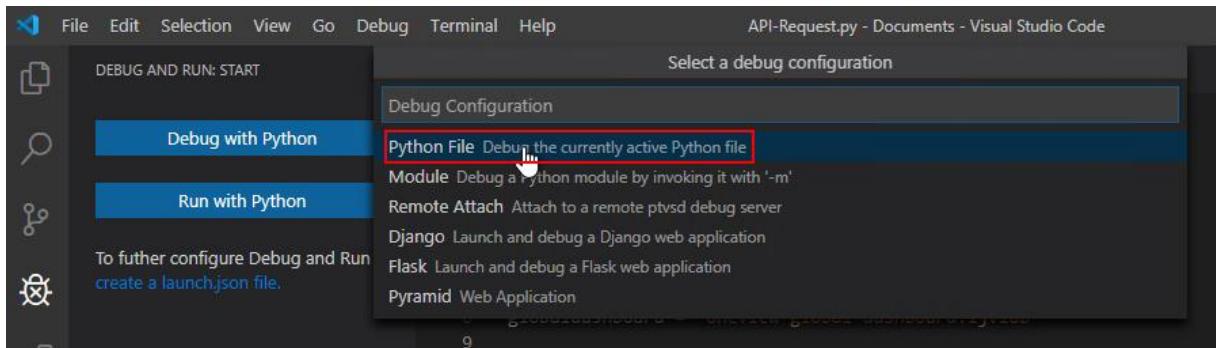


- a. Click on **Create a launch.json file** to configure Debug and Run





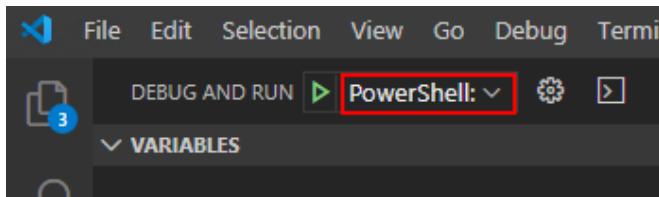
- b. Select the first option **Python File**:



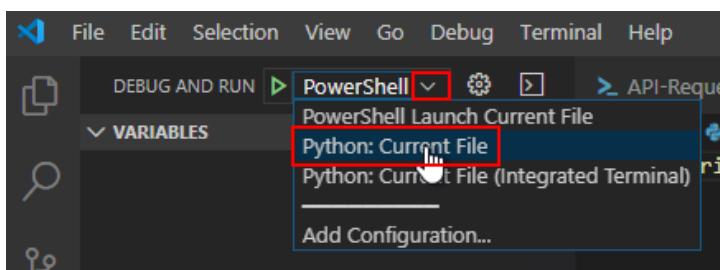
- c. Press **CTRL + S** to save the configuration file

- d. Press **CTRL + F4** to close the file

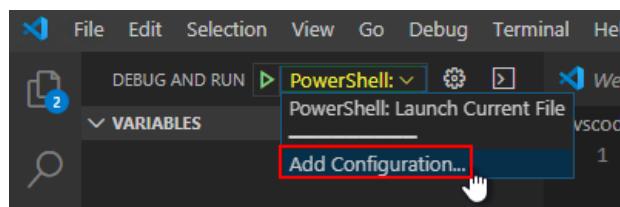
3. If PowerShell is displayed in the Debug and Run dialog:



- a. Select **Python: Current File** from the Debug and Run drop-down menu and you are done.

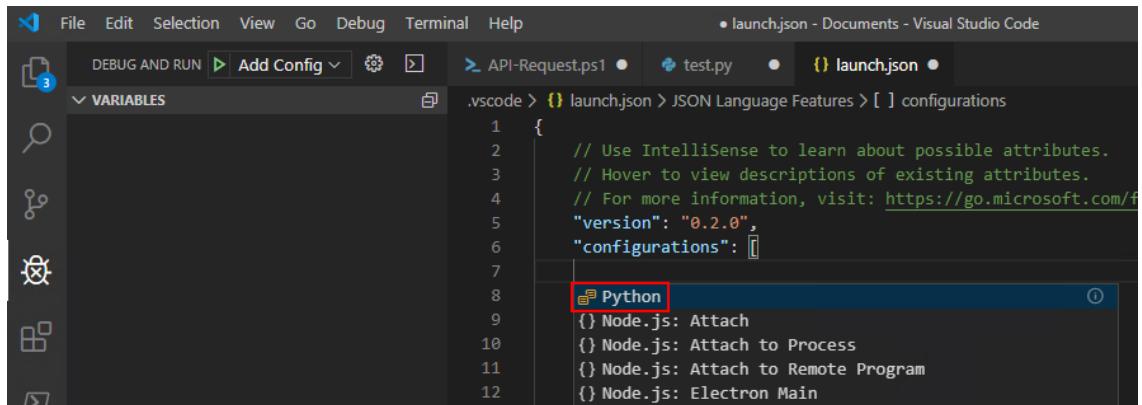


- i. If there is no Python configuration, you need to select **Add configuration** from the drop-down menu:

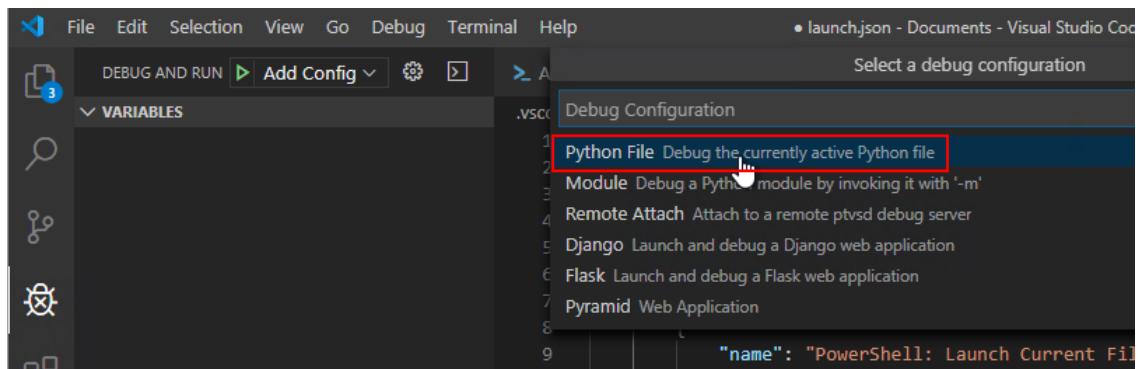




ii. Select **Python**



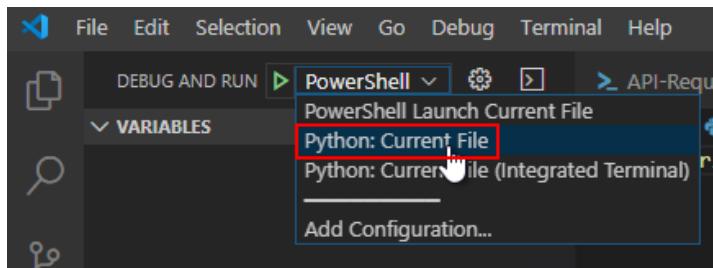
iii. Then select **Python File**



iv. Press **CTRL + S** to save the configuration file

v. Press **CTRL + F4** to close the file

vi. Then select **Python: Current File** from the Debug and Run drop-down menu



The Visual Studio Code debug and run setup is now complete, we can start debugging our script.

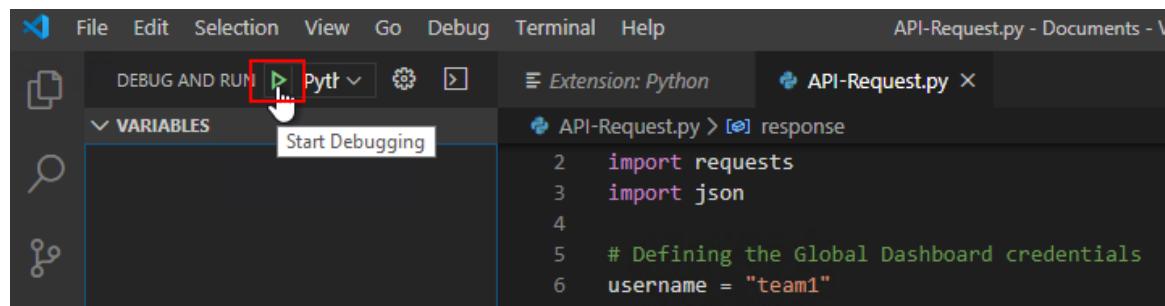


Using your mouse, drop a Breakpoint on the `print(response)` line. Breakpoints are normally shown as red filled circles, can be toggled by clicking on the editor margin:

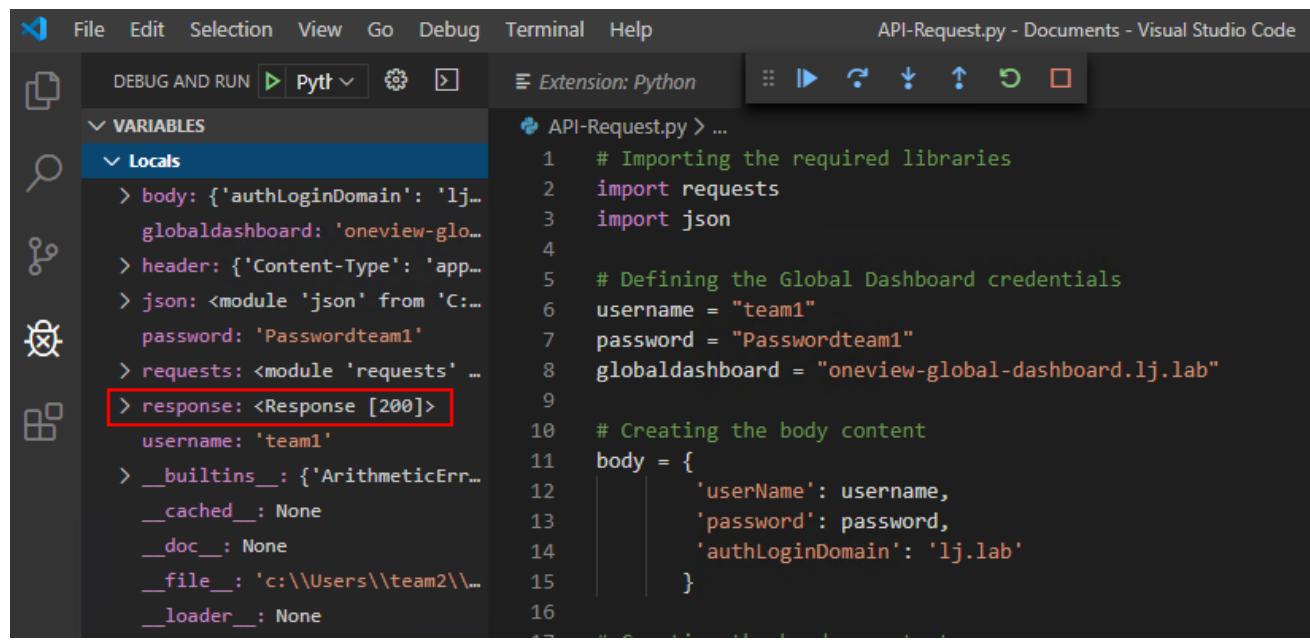
```
22
23     # Creating the web request with body and header content
24     response = requests.post(
25         url="https://" + globaldashboard + "/rest/login-sessions",
26         verify=False,
27         json=body,
28         headers=header
29     )
30
31     print(response)
32
```



Then click on **Start Debugging** to run the script



On the left-hand pane, the variables content is displayed, a successful API request response code **200** should be displayed for the `response` variable:



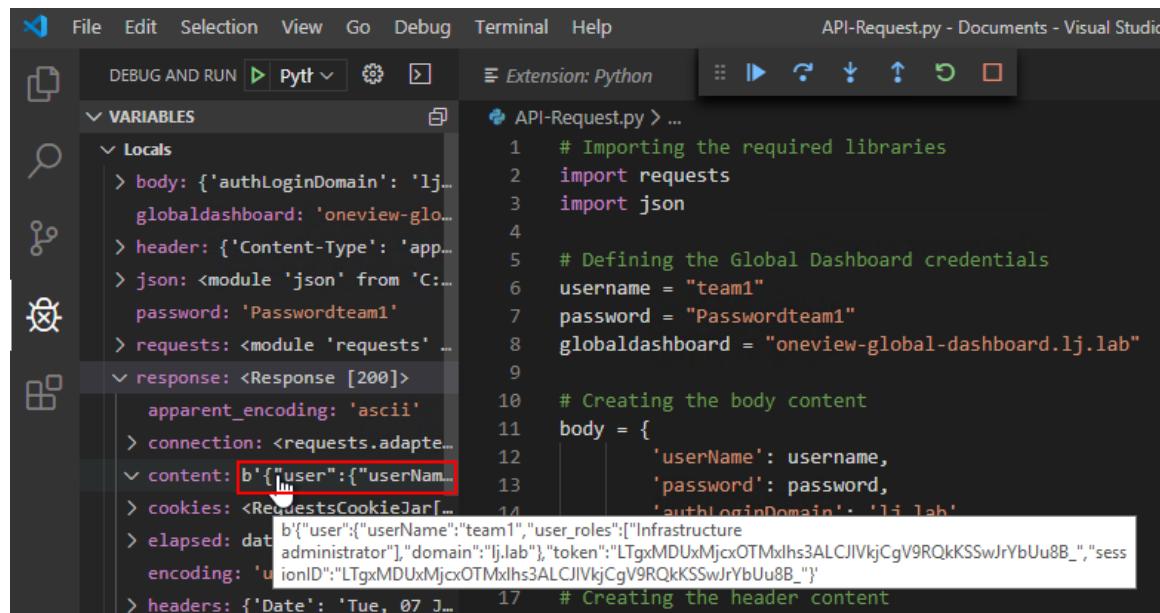


`response` is the variable we are using to store the result of `requests.post`

You can discover the content of `response` by using the expand icon:

```
password: 'Passwordteam1'  
> requests: <module 'requests' ...  
> response: <Response [200]>  
  username: 'team1'  
> __builtins__: {'ArithmeticErr...  
  __cached__: None
```

If you hover your mouse over the `content` description you will get the following popup box:



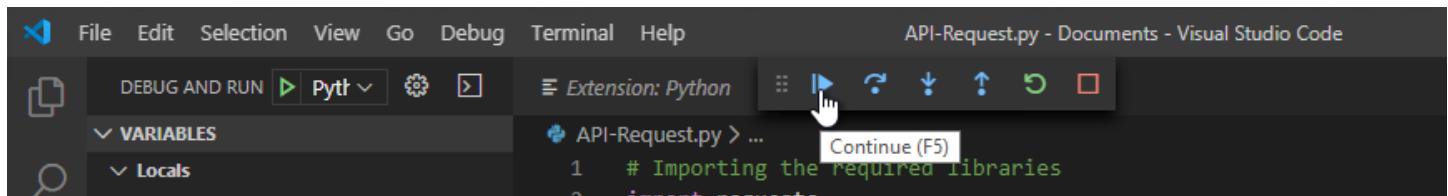
This is the response received from the API

```
b'{"user":{"user_name": "team1", "user_roles": ["Infrastructure administrator"], "domain": "lj.lab"}, "token": "LTgxMDUxMjcxOTMxLhs3ALCJIVkjCgV9RQkKSSwJrYbUu8B_"}'
```

Notice that the content of `content` is in JSON format. So, to access to the `sessionID`, we need to convert the response into a Python dictionary, easier to manipulate.



Click on **Continue** or **F5** to continue the script execution.



Then you can remove the breakpoint by directly clicking on the breakpoint or by using the menu **Debug / Remove all Breakpoints**.

So, to convert the JSON response into a Python dictionary, we are going to use the `json` library, imported at the beginning of our script. A Python dictionary is a mapping of unique keys to values that can be easily manipulated and accessed.

Add the following lines at the end of the script:

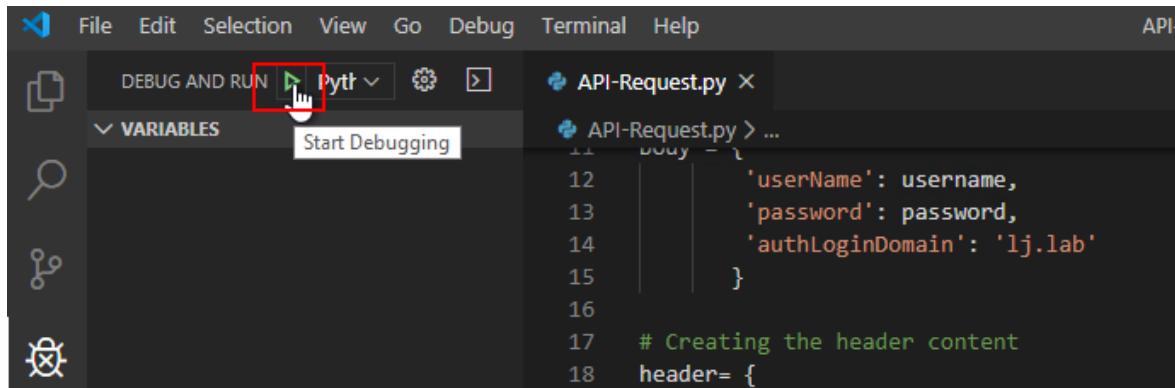
```
loginsessionresponse = json.loads(response.content)
print(json.dumps(loginsessionresponse, indent=4))
```

The first line converts the content of `response.content` into a dictionary. The second line is used to display `loginsessionresponse` in a nice format on the console:

```
23 # Creating the web request with body and header content
24 response = requests.post(
25     url="https://" + globaldashboard + "/rest/login-sessions",
26     verify=False,
27     json=body,
28     headers=header
29 )
30
31 print(response)
32
33 loginsessionresponse = json.loads(response.content)
34
35 print(json.dumps(loginsessionresponse, indent=4))
36
```



Then press **Start Debugging** to run the script again with the two new lines



```
File Edit Selection View Go Debug Terminal Help API-  
DEBUG AND RUN Python Variables API-Request.py x  
VARIABLES Start Debugging  
API-Request.py > ...  
body -  
12     'userName': username,  
13     'password': password,  
14     'authLoginDomain': 'lj.lab'  
15   }  
16  
17   # Creating the header content  
18   header = {
```

On the terminal window, you should see the content of `loginsessionresponse` structured as a Python dictionary:

```
{  
    "user": {  
        "userName": "team1",  
        "user_roles": [  
            "Infrastructure administrator"  
        ],  
        "domain": "lj.lab"  
    },  
    "token": "MzYxMTIyNDg4Mjk5wS3BffDY8XrxTlstG1uEwgQ-qFpo070T",  
    "sessionID": "MzYxMTIyNDg4Mjk5wS3BffDY8XrxTlstG1uEwgQ-qFpo070T"  
}  
PS C:\Users\team2\Documents>
```

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 use `loginsessionresponse['sessionID']`

So, add the next line in your script to save the Global Dashboard session ID into the variable `sessionID`:

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

Now every Global Dashboard API call method we do will use this `sessionID` token for the authentication.



Let's for instance get all resource alerts in Global Dashboard, we can add the following lines:

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

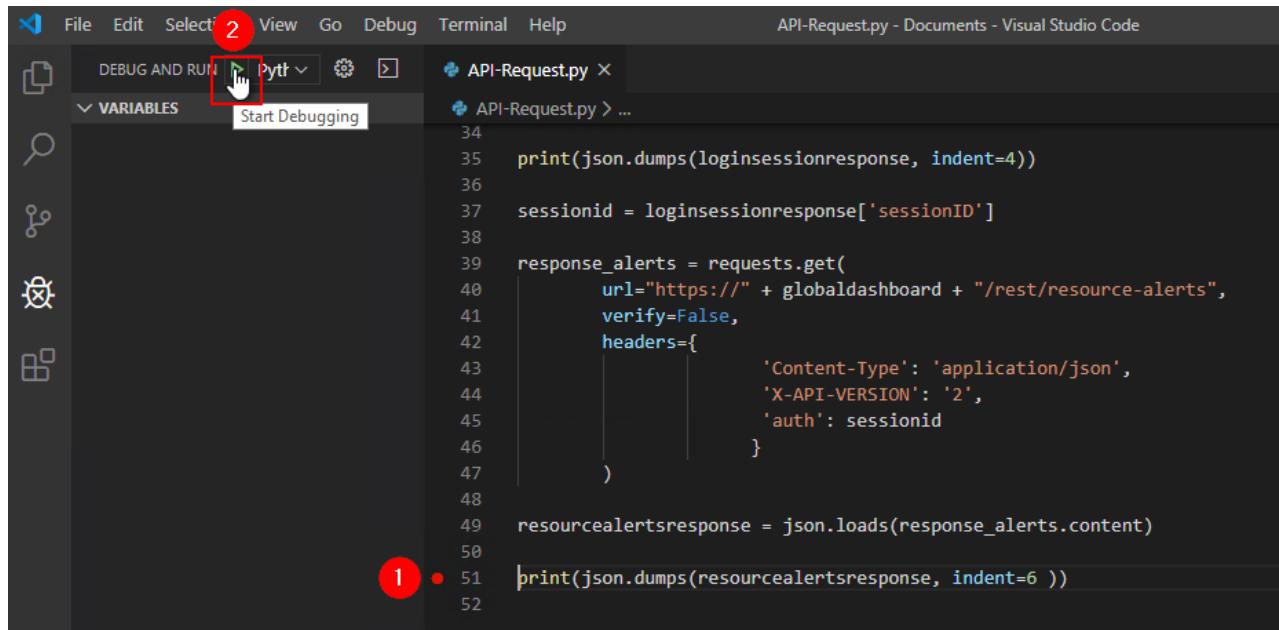
We use the variable `sessionid` to store the value of the authentication token. Then we use it in the header with `auth`. We also use the `verify=False` option to avoid a certificate verification failure as we are using a self-signed certificate in Global Dashboard.

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

Then to display the list of alerts, we can use the same previous `json()` and `print()` methods:

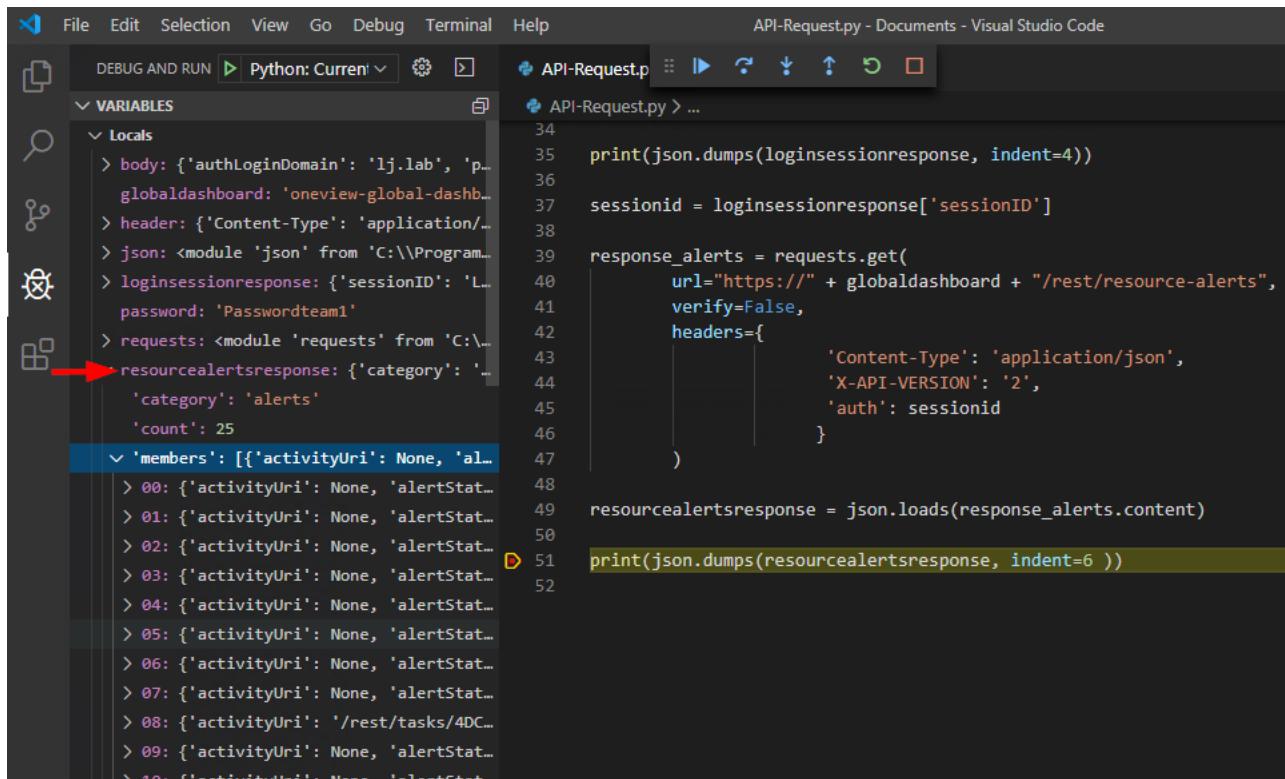
```
resourcealertsresponse = json.loads(response_alerts.content)  
print(json.dumps(resourcealertsresponse, indent=6))
```

Now put a breakpoint on the last line and click **Run**





Then look for the `resourcealertsresponse` variable in the left-hand debug pane to understand its content:



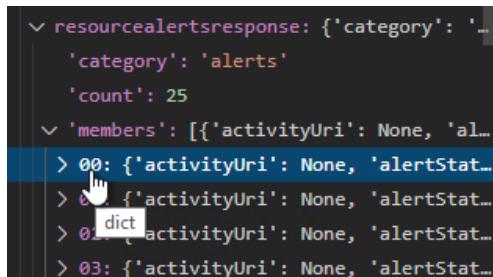
```
File Edit Selection View Go Debug Terminal Help API-Request.py - Documents - Visual Studio Code DEBUG AND RUN Python: Current API-Request.py ... 34 35 print(json.dumps(loginsessionresponse, indent=4)) 36 37 sessionid = loginsessionresponse['sessionID'] 38 39 response_alerts = requests.get( 40     url="https://" + globaldashboard + "/rest/resource-alerts", 41     verify=False, 42     headers={ 43         'Content-Type': 'application/json', 44         'X-API-VERSION': '2', 45         'auth': sessionid 46     } 47 ) 48 49 resourcealertsresponse = json.loads(response_alerts.content) 50 51 52
```

We see that the alerts are grouped into a list, sometimes referred as a collection. This list is called *members*.



```
resourcealertsresponse: {'category': ..., 'category': 'alerts', 'count': 25} members: [{} activityUri: None, 'alertStat... 00: {} activityUri: None, 'alertStat... 01: {} activityUri: None, 'alertStat... 02: {} activityUri: None, 'alertStat...
```

In this list, we have several dictionaries (25), each representing an alert.



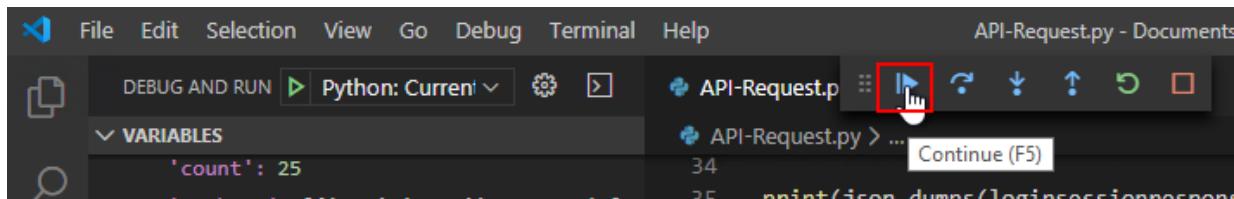
```
resourcealertsresponse: {'category': ..., 'category': 'alerts', 'count': 25} members: [{} activityUri: None, 'alertStat... 00: {} activityUri: None, 'alertStat... 01: {} activityUri: None, 'alertStat... 02: {} dict activityUri: None, 'alertStat...
```



Then each dictionary is represented by a paired list of "items": "value"

```
  ↴ 'members': [{activityUri': None, 'al...
  ↴ 00: {'activityUri': None, 'alertStat...
    'activityUri': None
    'alertState': 'Locked'
    'alertTypeID': 'profilemgr.Connecti...
    'applianceLocation': '192.168.1.110'
    'applianceModel': 'Synergy Composer'
    'applianceName': 'composer'
    'applianceVersion': '5.00.01-041026...
    'appluri': '/rest/appliances/ec4638...
    'assignedToUser': None
  > 'associatedEventUris': ['/rest/even...
  > 'associatedResource': {'association...
    'associatedResourceName': 'win-1'
    'associatedResourceOriginalUri': '...
    'category': 'alerts'
  > 'changeLog': []
    'clearedByUser': None
```

Click now on **Continue**





The alerts are displayed on the terminal window as requested in the script. Now scroll up to the beginning of the alert

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Python Debug Consc + □ □ ^ X

warnings.warn(
PS C:\Users\team2\Documents> cd 'c:\Users\team2\Documents'; ${env:PYTHONIOENCODING}='UTF-8';
UFFERED}=1'; & 'C:\Program Files\Python38\python.exe' 'c:\Users\team2\.vscode\extensions\ms
019.11.50794\pythonFiles\ptvsd_launcher.py' '--default' '--client' '--host' 'localhost' '--p
\Users\team2\Documents\API-Request.py'
C:\Program Files\Python38\lib\site-packages\urllib3\connectionpool.py:997: InsecureRequestWa
d HTTPS request is being made. Adding certificate verification is strongly advised. See: ht
thedocs.io/en/latest/advanced-usage.html#ssl-warnings
    warnings.warn(
<Response [200]>
{
    "user": {
        "userName": "team1",
        "user_roles": [
            "Infrastructure administrator"
        ],
        "domain": "lj.lab"
    },
    "token": "LTgwMTk3MTQ5ODEzUSQtI5_wddXeBwFbhNTndki9-SS0M8Kz",
    "sessionID": "LTgwMTk3MTQ5ODEzUSQtI5_wddXeBwFbhNTndki9-SS0M8Kz"
}
C:\Program Files\Python38\lib\site-packages\urllib3\connectionpool.py:997: InsecureRequestWa
ifi
C:\Program Files\Python38\lib\site-packages\urllib3\connectionpool.py:997: InsecureRequestWarning:
Unverified HTTPS request is being made. Adding certificate verification is strongly advised. See: h
ttps://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
    warnings.warn(
{
    "category": "alerts",
    "count": 25,
    "members": []
    {
        1
        2
        "type": "AlertResourceV3",
        "uri": "/rest/resource-alerts/d0c6de33-d008-40ae-95b7-1537cb2a9465",
        "category": "alerts",
        "eTag": "2020-01-03T17:31:46.264Z",
        "created": "2020-01-03T17:31:46.264Z",
        "modified": "2020-01-03T17:31:46.264Z",
```

The list here is identified by the [...] (1) and the dictionary by the {...} (2)

So, we can reduce the result to the alerts only by printing only the *memberlist*:

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



And finally, to display the number of alerts and the alerts in a nice format, you can start by removing all `print()` functions (or commenting each line with a `#`) in the script and enter the following commands:

```
print("\n\nThe number of alerts is " + str(len(alerts)) + ":")

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 must copy/paste each line of the loop at a time.

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

This loop is used to display all alerts in the list but using only the `severity`, `description` and `correctiveAction` fields as illustrated below:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: Python Debug Consc + □

The number of alerts is 25:

Severity: Critical
Description: An error has occurred on connection 8. Interconnect {"name":"Frame2, interconnect 6","uri":"/rest/interconnects/4b9"} port 25 subport b is unlinked.
Corrective Action: Verify that downlink port is enabled and link status is linked. Verify that the interconnect link topology is configured state and there are no network connectivity issues between OneView and the interconnect. If the problem persists, contact HPE and provide them with a support dump.

Severity: Critical
Description: An error has occurred on connection 6. Interconnect {"name":"Frame2, interconnect 6","uri":"/rest/interconnects/4b9"} port 25 subport d is unlinked.
Corrective Action: Verify that downlink port is enabled and link status is linked. Verify that the interconnect link topology is configured state and there are no network connectivity issues between OneView and the interconnect. If the problem persists, contact HPE and provide them with a support dump.

Severity: Critical
Description: Connection on downlink port 25, subport b has failed. The subport is unlinked.
Corrective Action: Verify that the downlink port is enabled and link status is linked. Verify that there are no critical interconnect {"name":"Frame2, interconnect 6","uri":"/rest/interconnects/4b94ebc4-5bc3-447b-9b6b-66b3f9761bb7"}. If a critical interconnect {"name":"Frame2, interconnect 6","uri":"/rest/interconnects/4b94ebc4-5bc3-447b-9b6b-66b3f9761bb7"} is a critical issue, follow the resolution steps in the alert to restore the interconnect link topology. Verify that the interconnect is in a connected state and there are no network connectivity problems between OneView and the interconnect. If the problem persists, contact your authorized support representative.

Severity: Critical
Description: Connection on downlink port 25, subport d has failed. The subport is unlinked.
Corrective Action: Verify that the downlink port is enabled and link status is linked. Verify that there are no critical interconnect {"name":"Frame2, interconnect 6","uri":"/rest/interconnects/4b94ebc4-5bc3-447b-9b6b-66b3f9761bb7"}. If a critical interconnect {"name":"Frame2, interconnect 6","uri":"/rest/interconnects/4b94ebc4-5bc3-447b-9b6b-66b3f9761bb7"} is a critical issue, follow the resolution steps in the alert to restore the interconnect link topology. Verify that the interconnect is in a connected state and there are no network connectivity problems between OneView and the interconnect. If the problem persists, contact your authorized support representative.
```

Note: `members` is a list so we can get the first alert by entering `[0]`:

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

or `[-1]` to get the last one:

```
print(json.dumps(alerts[-1], indent=6))
```



Just for information

One of the cool features available in Postman is the Generate code snippets. This feature allows you generate snippets of code in various languages and frameworks, including Python.

So, after having built an API request, you can simply click on **Code** under the blue Send button to open the Generate code snippets modal.

The screenshot shows the Postman interface with a GET request to `https://composer.lj.lab/rest/ethernet-networks`. The Headers tab is selected, showing two entries: `x-api-version` with value `1200` and `auth` with a long token value. The **Code** button in the top right corner is highlighted with a red box.

Use the dropdown menu to select a language - some languages have multiple options. This lets you select different frameworks from which to make your request.

The screenshot shows the 'Generate Code Snippets' modal. The left sidebar lists various languages: HTTP, C (LibCurl), cURL, C# (RestSharp), Go, Java, JavaScript, NodeJS, Objective-C (NSURL), OCaml (Cohttp), PHP, Python, Ruby (NET::Http), Shell, and Swift (NSURL). The Python section is expanded, showing options for `http.client (Python 3)` and `Requests`. The `Requests` option is highlighted with a red box and has a cursor icon over it. A 'Copy to Clipboard' button is visible at the top right of the modal.



This is an example of the Python code generated for a get /rest/ethernet-networks call:

GENERATE CODE SNIPPETS X

Python Requests ▾ Copy to Clipboard

```
1 import requests
2
3 url = "https://composer.1j.lab/rest/ethernet-networks"
4
5 payload = ""
6 headers = {
7     'x-api-version': "1200",
8     'auth': "MjA4NzU4ODc3OEE5gs788qpTyvv-azg6LmonJ3R5VRoCKX89",
9     'cache-control': "no-cache",
10    'Postman-Token': "ab96bbb-cda5-4422-b933-8ff30563a673"
11 }
12
13 response = requests.request("GET", url, data=payload, headers=headers)
14
15 print(response.text)
```

This code looks familiar, doesn't it?

This concludes the Python Lab.



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>