2-RedfishAnsibleUsingBuiltinUri

March 15, 2021

1 Use of the Ansible built-in URI method to query a Redfish service

Version 0.155

1.1 Introduction

This Jupyter Notebook uses the Ansible built-in uri method and an authentication token for getting and setting parameters of an iLO 5 Redfish service.

The use case of this Notebook is to explain how you can get and set the properties of a OneView managed compute node without any additional iLO user than the built-in `_HPOneViewAdmin` user.

In addition you will learn the basic methodology to discover Redfish resources instead of assuming their final location in the Redfish tree.

1.2 Environment preparation

The following cell sets environment variables and checks the connectivity to your Synegy iLO 5 simulator.

[1]:
```
########## Environment preparation (Version: 0.132) #############

# Set Student ID number
stdid=825
Id=$(id --user --name)
NbId=2
InvFile=${NbId}/hosts

# location and ports variables
IloSyBasePort=46000
let iLO5SimulatorBasePort=$IloSyBasePort
let iLO5SimulatorPort=${iLO5SimulatorBasePort}+${stdid}

iLO5SimulatorIP=ilo5simulators
iLOSimulator=${iLO5SimulatorIP}:${iLO5SimulatorPort}
iLO5SimulatorURI=https://${iLOSimulator}

# Fake Credentials as we are testing against a BMC simulator
```

```
OvSsoToken="FakeOvSsoToken"

# Miscellaneous
WorkshopDir=$PWD
HpePythonRedfishVenv="${NbId}/HpePythonRedfishVenv"
export PYTHONPATH="${WorkshopDir}/${NbId}/library/"
w=$(basename $PWD)

alias ResetSimulators="../create-globalbmc.shc.x &>/dev/null ; sleep 1"

# Verify we can reach the remote Bmcs on the right HTTPS ports.
for bmc in iLO5Simulator ; do
    ip="${bmc}IP" ; port=$(echo ${bmc}Port)
    nc -vz  $(eval echo "\$${ip}") $(eval echo "\$${port}") &>/dev/null &&
        echo -e "\n\tGood News: $bmc is reachable" \
        || echo "WARNING: Problem reaching $bmc"
done


# Create the Ansible inventory file
cat > ${InvFile} << __EOF__
[OneViewManagedBmcs]
${iLO5SimulatorIP} ansible_port=${iLO5SimulatorPort}

[OneViewManagedBmcs:vars]
ansible_python_interpreter=${WorkshopDir}/${HpePythonRedfishVenv}/bin/python3
ansible_search_path=${HpePythonRedfishVenv}
# Below is a fake session token as we are testing against an iLO 5 simulator
token="${OvSsoToken}"
__EOF__
```

```
        Good News: iLO5Simulator is reachable
```

1.3 Virtual Python environment creation

In order to completely isolate this notebook environment from other notebooks or student python environments, it is safer to create your dedicated Python virtual environment.

> NOTE: This `Venv` creation can take up to **2 minutes**. Just wait until the message `Finished creating Venv` is displayed

[2]:
```
# Create Virtual Python environment (Venv)
[ -d ${HpePythonRedfishVenv} ] && rm -r ${HpePythonRedfishVenv} &>/dev/null
python3 -m venv ${HpePythonRedfishVenv}                         &>/dev/null
source ${HpePythonRedfishVenv}/bin/activate                     &>/dev/null
export PS1="[PEXP\[\]ECT_PROMPT>"                               # Avoid Venv␣
 ↪long prompt messing up outputs
```

```
# Install latest Ansible in the Venv
pip install jmespath                                                    &>/dev/null
pip install ansible                                                     &>/dev/null

echo -e "\n\n\tFinished creating Venv\n\n"
```

```
(HpePythonRedfishVenv)

        Finished creating Venv
```

1.3.1 Restart iLO 5 simulator

If you need or desire to restart your iLO 5 simulator to restart this workshop from scratch or for other reasons, run the following cell at any time.

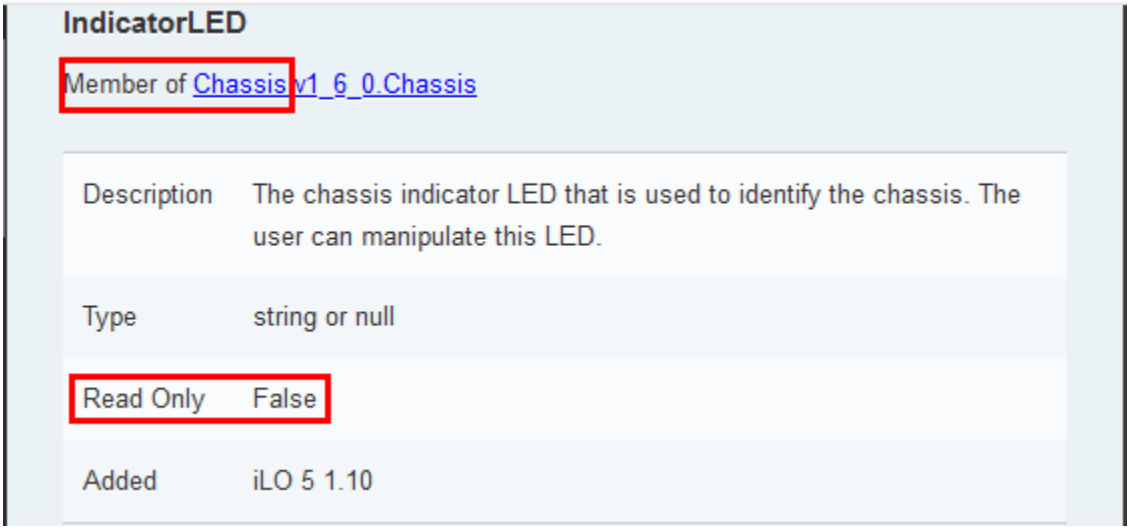
[3]:
```
# iLO 5 Simulator restart  (Version 0.1)
ResetSimulators

# Verify we can reach the remote Bmcs on the right HTTPS ports.
for bmc in iLO5Simulator ; do
    ip="${bmc}IP" ; port=$(echo ${bmc}Port)
    nc -vz  $(eval echo "\$${ip}") $(eval echo "\$${port}") &> /dev/null &&
        echo "$bmc is reachable" \
        || echo "WARNING: Problem reaching $bmc"
done
```

```
iLO5Simulator is reachable
```

1.4 Get and Set Redfish properties using Ansible built-in uri module

In this section you will change the status of the Unit Identification light (UID, also called `IndicatorLED` in Redfish terminology) of compute nodes in order to facilitate their location in the Datacenter before maintenance. To make this exercise more realistic, you will automatically modify the enclosure UID/LED of these compute nodes if they are part of an enclosure/frame infrastructure.

The `IndicatorLED` resource location is standardized by Redfish as part of the `Chassis` data type and documented in the HPE Redfish API reference document:



IndicatorLED

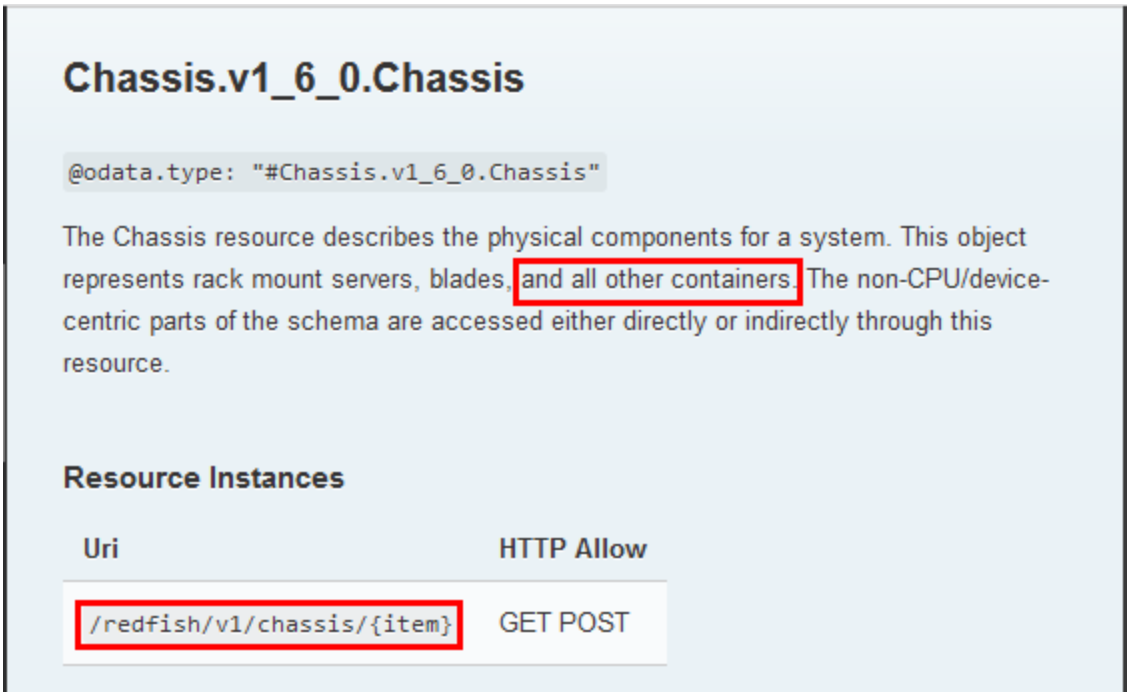
Member of Chassis.v1_6_0.Chassis

Description	The chassis indicator LED that is used to identify the chassis. The user can manipulate this LED.
Type	string or null
Read Only	False
Added	iLO 5 1.10

The `Chassis` data type, as per Redfish, is located at `/redfish/v1/Chassis`. From this entry point you will retrieve the `{item}` list of each chassis composing this data type.

In a Synergy compute node, this list is composed of two chassis: a chassis called `enclosurechassis` containing the properties of the frame enclosure, and a chassis called `1` for the compute node.

> **NOTE**: The chassis names are not standadized by Redfish and may change over time. Moreover, this naming convention is definitively different for Moonshot, Superdome and other vendors of blade computers. Hence, if you want your script to work against other Redfish implementations than ilO based servers, you need to discover each `{item}` in the `Chassis` collection instead of assuming it.



Chassis.v1_6_0.Chassis

```
@odata.type: "#Chassis.v1_6_0.Chassis"
```

The Chassis resource describes the physical components for a system. This object represents rack mount servers, blades, and all other containers. The non-CPU/device-centric parts of the schema are accessed either directly or indirectly through this resource.

Resource Instances

Uri	HTTP Allow
/redfish/v1/chassis/{item}	GET POST

1.4.1 Show, modify and verify a Redfish property using Ansible uri

The following cell discovers the `Chassis` collection of the nodes listed in the hosts inventory file created in the first cell of this notebook. Then, it prints selected properties of each member of the collection, including the location and the value of the `IndicatorLED`.

The next tasks toggle the `IndicatorLED` value (`Off` - `Lit`) and apply the modification on each item of the collection.

The last part of the playbook validates the modification by retrieving again the `IndicatorLED` property of each item of the chassis collection.

All of the above is performed using the ansible.builtin.uri module and a (fake) OneView SSO token obtained in the previous notebook.

A convenient way to study the playbook of the next cell is to open it in a different view in this pane. Right click on this Notebook tab name and select `New View for Notebook` to open a new view:

Then, click on this file link.

If you need more space, type `Ctrl-B` (or `Command-B` on a Mac) to hide the left pane. You can make it reappear by hitting `Ctrl-B` again.

[4]:
```
# Modify IndicatorLED(s) using the Ansible built-in URI module
ansible-playbook -i ${InvFile} ${NbId}/SetIndicatorLEDUsingBuiltInUri.yml
```

```
PLAY [OneViewManagedBmcs] ******************************************************

TASK [1.0- Discover chassis collection in standard root service
/redfish/v1/Chassis] ***
ok: [ilo5simulators]

TASK [1.1- Save location of each item in the collection] ***********************
ok: [ilo5simulators]

TASK [1.2- Retrieve and print selected properties of each item of the
collection] ***
ok: [ilo5simulators] => (item=/redfish/v1/Chassis/1/)
ok: [ilo5simulators] => (item=/redfish/v1/Chassis/enclosurechassis/)

TASK [debug] *******************************************************************
ok: [ilo5simulators] => {
    "msg": [
        {
            "ChassisType": "Blade",
            "Id": "1",
            "IndicatorLED": "Off"
        },
        {
```

```
            "ChassisType": "Enclosure",
            "Id": "enclosurechassis",
            "IndicatorLED": "Off"
        }
    ]
}

TASK [2.1- PATCH IndicatorLED with new value using Ansible built-in uri module]
***
ok: [ilo5simulators] => (item={'Id': '1', 'IndicatorLED': 'Off'})
ok: [ilo5simulators] => (item={'Id': 'enclosurechassis', 'IndicatorLED': 'Off'})

TASK [3.0- Retrieve IndicatorLED New status to verify previous PATCH] **********
ok: [ilo5simulators] => (item=/redfish/v1/Chassis/1/)
ok: [ilo5simulators] => (item=/redfish/v1/Chassis/enclosurechassis/)

TASK [debug] ********************************************************************
ok: [ilo5simulators] => {
    "msg": [
        {
            "ChassisType": "Blade",
            "Id": "/redfish/v1/Chassis/1/",
            "IndicatorLED": "Lit"
        },
        {
            "ChassisType": "Enclosure",
            "Id": "/redfish/v1/Chassis/enclosurechassis/",
            "IndicatorLED": "Lit"
        }
    ]
}

PLAY RECAP **********************************************************************
ilo5simulators             : ok=7    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

1.4.2 Test the same playbook against a rack-mount server

The following cell switches your environment toward an **HPE DL360 Gen10** simulator and then runs again the Ansible Playbook. You will notice that the same playbook works for both a Synergy compute node and a rack-mount server although it is not enclosed in any frame or enclosure chassis.

> **NOTE**: In a real and physical environment, session token authentication against HPE iLO 5 rack mount servers is supported when managed by a OneView appliance. If not managed by OneView, you have to modify the playbook code and supply `url_username` and `url_password` parameters to the `uri` method for basic authentication. Or you could create an additional task with basic authentication to create a session and then extract

a session token from the headers of the response of the Redfish service. This mechanism is deeply explained in the Redfish API 101 Workshop-on-Demand .

[5]:
```
# location and ports variables
IloDlBasePort=45000
let iLO5SimulatorBasePort=$IloDlBasePort
let iLO5SimulatorPort=${iLO5SimulatorBasePort}+${stdid}

iLO5SimulatorIP=ilo5simulators
iLOSimulator=${iLO5SimulatorIP}:${iLO5SimulatorPort}
iLO5SimulatorURI=https://${iLOSimulator}

# Adapt the Ansible inventory file
cat > ${InvFile} << __EOF__
[OneViewManagedBmcs]
${iLO5SimulatorIP} ansible_port=${iLO5SimulatorPort}

[OneViewManagedBmcs:vars]
ansible_python_interpreter=${WorkshopDir}/${HpePythonRedfishVenv}/bin/python3
ansible_search_path=${HpePythonRedfishVenv}
# Below is a fake session token as we are testing against an iLO 5 simulator
token="${OvSsoToken}"
__EOF__


# Modify IndicatorLED(s) using the Ansible built-in URI module against an HPE␣
 ↪DL360 Gen10 ilo5
ansible-playbook -i ${InvFile} ${NbId}/SetIndicatorLEDUsingBuiltInUri.yml
```

```
PLAY [OneViewManagedBmcs] ******************************************************

TASK [1.0- Discover chassis collection in standard root service
/redfish/v1/Chassis] ***
ok: [ilo5simulators]

TASK [1.1- Save location of each item in the collection] ***********************
ok: [ilo5simulators]

TASK [1.2- Retrieve and print selected properties of each item of the
collection] ***
ok: [ilo5simulators] => (item=/redfish/v1/Chassis/1/)

TASK [debug] *******************************************************************
ok: [ilo5simulators] => {
    "msg": [
        {
            "ChassisType": "RackMount",
```

```
            "Id": "1",
            "IndicatorLED": "Off"
        }
    ]
}

TASK [2.1- PATCH IndicatorLED with new value using Ansible built-in uri module]
***
ok: [ilo5simulators] => (item={'Id': '1', 'IndicatorLED': 'Off'})

TASK [3.0- Retrieve IndicatorLED New status to verify previous PATCH] **********
ok: [ilo5simulators] => (item=/redfish/v1/Chassis/1/)

TASK [debug] *******************************************************************
ok: [ilo5simulators] => {
    "msg": [
        {
            "ChassisType": "RackMount",
            "Id": "/redfish/v1/Chassis/1/",
            "IndicatorLED": "Lit"
        }
    ]
}

PLAY RECAP *********************************************************************
ilo5simulators             : ok=7    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

1.5 Summary

In this workshop, you used the Ansible built-in `uri` module to get and set Redfish properties, using a OneView Single Sign On (fake) token and without assuming their location. You validated the same code against two different types of server, proving its portability. Crawling the Redfish tree using `yaml` is possible but may become very quickly complex for managing resources deeper than the second level of the Redfish tree.

Select the next Notebook to perform the same exercise with the HPE provided playbooks.