1-Aspire-RedfishBash

May 15, 2020

1 Hack Shack @Aspire

Powered by HPE DEV Team

1.0.1 Speakers : Troy Heber / François Donzé

1.0.2 Redfish REST API overview

Version 0.58

1.1 Introduction

This Jupyter notebook defines environment variables that will be used through the rest of the notebook. Then, it explains the Redfish resource tree as well as how to use the session authentication mechanism using Bash and the cURL tool against an OpenBMC simulator. Then, it performs a reset of the OpenBMC. For didactic reasons, commands presented in this notebook are not optimized.

1.2 Create environment variables

The following `bash` code defines environment variables (i.e. IP address, username, password....) depending on your student ID number stored in variable `$Stud`. It creates as well a `.json` file containing the credentials of your OpenBMC appliance required to open a Redfish session.

Click in the cell below and then click on the icon to create the environment variables and the json file.

[1]:
```
# Set Student ID number
Stud="00"
echo "You are Student $Stud"

# Create BMC location/ports variables
BmcIP=openbmcs:443${Stud}          # OpenBMC simulator
BmcURI="https://${BmcIP}"

# BMC Administrator credentials
BmcUser="student"
BmcPassword='P@ssw0rd!'

# Minimum required Redfish headers
HeaderODataVersion="OData-Version: 4.0"
HeaderContentType="Content-Type: application/json"

# Data files
ResponseHeaders="ResponseHeaders.txt"   # Used to hold HTTP response headers
SessionData="./CreateSession-data.json" # Body/Workload used to create the␣
 ↪Redfish session
cat > ${SessionData} << __EOF__
{
        "UserName": "$BmcUser",
        "Password": "$BmcPassword"
}
__EOF__
```

```
You are Student 00
```

1.3 Retrieve the Redfish Service Entry point content (Root)

The Redfish Service Entry point is **/redfish/v{RedfishVersion}/**.

Run the next cell to retrieve Redfish version(s) available today in your OpenBMC.

This request does not require any authentication.

If you are not familiar with cURL, get help from its manual page.

[2]:
```
echo "Available Redfish version(s):"

curl --insecure --noproxy "localhost, 127.0.0.1" --silent \
  --header  "$HeaderContentType" --header "$HeaderODataVersion" \
  --request GET ${BmcURI}/redfish | jq
```

```
Available Redfish version(s):
```

```
{
  "v1": "/redfish/v1/"
}
```

The previous command returns only one available Redfish version implemented in your BMC: **`v1`**. Hence its Redfish Service Entry point is at **`/redfish/v1`**

It contains:

- Keys describing the **Root Service**: `@odata.context`, `Id`, `Name`, `RedfishVersion`, `UUID`....
- Services and collection URIs: `AccountService`, `Managers`, `Systems`.
- Links allowing direct access to resources beneath Root endpoints.

A `GET` of the Redfish Root content does not require authentication.

Run the following `curl` command to retrieve the Redfish Service Entry point content and take a look at the ouput.

Note that the keys are scattered among Services. For example, the `Id` key appears between the `Chassis` and `JsonSchemas` entry points. Other Redfish implementations (i.e. HPE iLO) may return a different output order.

Very important: All the objects present in this output are fully described in the Redfish `ServiceRoot` schema version `v1_5_0` as mentionned in `@odata.type`. You will learn how to browse the `JsonSchemas` resources later in this notebook.

The thing to remember for now is that a **single version of Redfish** holds multiple schemas versions. Then, resources can be added, moved or removed from a schema version to another one.

[3]:
```
echo "Redfish Service Entry point:"

curl --insecure --noproxy "localhost, 127.0.0.1" --silent \
  --header  "$HeaderContentType" --header "$HeaderODataVersion" \
  --request GET ${BmcURI}/redfish/v1 | jq
```

```
Redfish Service Entry point:
```

```
{
  "@odata.id": "/redfish/v1",
  "@odata.type":
"#ServiceRoot.v1_5_0.ServiceRoot",
  "AccountService": {
    "@odata.id":
"/redfish/v1/AccountService"
  },
  "CertificateService": {
    "@odata.id":
"/redfish/v1/CertificateService"
  },
  "Chassis": {
    "@odata.id":
"/redfish/v1/Chassis"
  },
  "Id": "RootService",
  "JsonSchemas": {
    "@odata.id":
"/redfish/v1/JsonSchemas"
  },
  "Links": {
    "Sessions": {
      "@odata.id":
"/redfish/v1/SessionService/Sessions"
    }
  },
  "Managers": {
    "@odata.id":
"/redfish/v1/Managers"
  },
  "Name": "Root Service",
  "RedfishVersion": "1.9.0",
  "Registries": {
    "@odata.id":
"/redfish/v1/Registries"
```

1.3.1 Resource map (highlights)

The above output lists the URI End Points holding all the resources for this Redfish version. Here is a light description of some of the most important ones:

- `AccountService`: collection of user accounts present in this BMC.
- `Chassis`: Collection of Chassis; "Physical view of the system" containing global physical asset info (i.e. power, thermal). A system can have multiple chassis (ex: HPE Superdome Flex).
- `Sessions`: Collection of current open sessions (ssh, https, GUI...).
- `Managers`: Collection of BMCs. A server can have multiple BMCs (i.e. HPE Moonshot have one iLO pers SOC).
- `Systems`: Collection of Systems; "Logical view of the server" with resources like Model, Serial number, Boot Order, NIC MAC, BIOS parameters ... A server can have multiple Systems (i.e. HPE Superdome Flex can have multiple hardware partitions: one per RedfishSystem).

1.4 Create a Redfish session using cURL

All the URIs below the Root entry point requires authentication. In this section you'll go through the session authentication method as it may differ from other Rest APIs (i.e. OneView).

The following `curl` command sends a `POST` request toward the standard `/redfish/v1/SessionService/Sessions` URI of your BMC. The body/workload of this request is in the `@${SessionData}` json file populated in the very first `bash` cell of this notebook (Environment variables). You can view its content by clicking on it from the left pane of your Jupyter environment.

Select and run the following cell.

If this `POST` request is successful, the BMC sends back a `Token` and a `Session Location` in the **headers of the response**. Response headers are saved in the `$ResponseHeaders` text file now present in the left pane of your Jupyter environment.

[4]:
```
echo 'Create Session and print body response:'

curl --dump-header  $ResponseHeaders \
     --insecure --noproxy "localhost, 127.0.0.1" --silent \
     --header "$HeaderContentType" --header "$HeaderODataVersion" \
     --request POST --data "@$SessionData" \
     ${BmcURI}/redfish/v1/SessionService/Sessions | jq
```

```
Create Session and print body response:
```

```
{
  "@odata.id":
"/redfish/v1/SessionService/Sessions/C8IJh1sHpg",
  "@odata.type":
"#Session.v1_0_2.Session",
  "Description": "Manager User
Session",
  "Id": "C8IJh1sHpg",
  "Name": "User Session",
  "UserName": "student"
}
```

1.5 Extract session token and session location

In the next cell, the `Session Token` (aka `Session Key`) and `Location` are retrieved from the headers of the BMC response and saved in variables. The `Session Token` will be used later to authenticate when needed. The `Sesion Location` will be used to close the session.

Run it.

```
[5]: BmcToken=$(awk '/X-Auth-Token/ {print $NF}' $ResponseHeaders | tr -d '\r')
BmcSessionLocation="$BmcURI"$(awk '/^Loca.*Se/ {gsub("https://.*/red", "/red",␣
↪$NF);print $NF}' $ResponseHeaders | tr -d '\r')

echo "Bmc Token: $BmcToken"
echo -e "Bmc Session Location: $BmcSessionLocation\n"
```

```
Bmc Token: vEnroek4cT6riWmQziQt
Bmc Session Location:
https://openbmcs:44300/redfish/v1/SessionService/Sessions/C8IJh1sHpg
```

1.6 Retrieve BMC parameters

Redfish locates BMC parameters under `/redfish/v1/Managers`. From there you'll be able to identify all the BMCs present in your server as well as their properties. Remember that computers like HPE Moonshot and HPE Superdome Flex can have several BMCs.

Your infrastructure is based upon servers with only one BMC. However we'll use a code suitable for servers with multiple BMCs.

The following cell lists the collection of all the BMCs present in your system. Since requests below the Redfish Root Entry point requires authentication, you must supply the **`X-Auth-Token`** header to `curl`.

Run it and note that there is only one BMC present in your OpenBMC appliance (`Member@odata.count = 1`). Note as well its location: `/redfish/v1/Managers/bmc`. Other Redfish implementations use different locations. For example, the URI of an HPE iLO in a ProLiant Server is `/redfish/v1/Managers/1`.

[6]:
```
echo "BMC collection:"
curl --insecure --silent --noproxy "localhost, 127.0.0.1"  \
  --header  "$HeaderContentType" --header "$HeaderODataVersion" \
  --header "X-Auth-Token: $BmcToken" \
  --request GET ${BmcURI}/redfish/v1/Managers | jq
```

```
BMC collection:
{
  "@odata.id":
"/redfish/v1/Managers",
  "@odata.type":
"#ManagerCollection.ManagerCollection",
  "Members": [
    {
      "@odata.id":
"/redfish/v1/Managers/bmc"
    }
  ],
  "Members@odata.count": 1,
  "Name": "Manager Collection"
}
```

The following cell extracts the name of the BMCs present in your system and then, for each BMC it extracts its properties.

Run it and review the properties returned by your OpenBMC. Among them you should notice the `Actions` and the `Oem` resources which need some explanation.

The `Actions` collection contains all the possible actions that can be performed on your BMC; With this version of OpenBMC, you can perform a reset of the BMC by posting the value `GracefulRestart` at `/redfish/v1/Managers/bmc/Actions/Manager.Reset`. You'll do this later.

The `Oem` collection contains resources specific to `OpenBmc` and not part of the Redfish standard. This is a way to allow computer makers to expose their specific and added value resources to the Rest API.

[7]:
```
BmcList=$(curl --insecure --silent --noproxy "localhost, 127.0.0.1"  \
  --header  "$HeaderContentType" --header "$HeaderODataVersion" \
  --header "X-Auth-Token: $BmcToken" \
  --request GET ${BmcURI}/redfish/v1/Managers | jq '.Members[]' | \
  awk -F/ '/@odata.id/ {print $NF}' | tr -d '"' )

echo "List of BMC(s) present in this system:"
echo -e "$BmcList\n"

for bmc in $BmcList ; do
    echo "Properties of BMC: $bmc"
    curl --insecure --noproxy "localhost, 127.0.0.1" --silent \
      --header  "$HeaderContentType" --header "$HeaderODataVersion" \
      --header "X-Auth-Token: $BmcToken" \
      --request GET ${BmcURI}/redfish/v1/Managers/${bmc} | jq
done
```

```
List of BMC(s) present in this system:
bmc

Properties of BMC: bmc
```

```
{
  "@odata.id":
"/redfish/v1/Managers/bmc",
  "@odata.type":
"#Manager.v1_3_0.Manager",
  "Actions": {
    "#Manager.Reset": {
      "ResetType@Redfish.AllowableValues": [
        "GracefulRestart"
      ],
      "target":
"/redfish/v1/Managers/bmc/Actions/Manager.Reset"
    }
  },
  "DateTime":
"2020-05-15T20:36:36+00:00",
  "Description": "Baseboard Management
Controller",
  "EthernetInterfaces": {
    "@odata.id":
"/redfish/v1/Managers/bmc/EthernetInterfaces"
  },
  "FirmwareVersion":
"2.8.0-dev-1427-g64e281927",
  "GraphicalConsole": {
    "ConnectTypesSupported": [
      "KVMIP"
    ],
    "MaxConcurrentSessions": 4,
    "ServiceEnabled": true
  },
  "Id": "bmc",
  "Links": {
    "ManagerForChassis": [
      {
        "@odata.id":
```

If you want to view the network protocols supported by your BMC, you can retrieve them with a `GET` of the `NetworkProtocol` URI mentionned in the output of the above `GET` request.

Run the next `curl`command. Its output should show an empty array of `NTPServers` (if not, contact your instructor). It contains as well the **Type** of the resources in this sub-tree: `@odata.type = #ManagerNetworkProtocol.v1_4_0.ManagerNetworkProtocol`.

Said differently, the `NetworkProtocol` resources falls in the **`ManagerNetworkProtocol`**. You will need this info later.

[8]:
```
echo "Network Protocol configuration:"
curl --insecure --silent --noproxy "localhost, 127.0.0.1"  \
  --header  "$HeaderContentType" --header "$HeaderODataVersion" \
  --header "X-Auth-Token: $BmcToken" \
  --request GET ${BmcURI}/redfish/v1/Managers/${bmc}/NetworkProtocol | jq
```

```
Network Protocol configuration:
```

```
{
  "@odata.id":
"/redfish/v1/Managers/bmc/NetworkProtocol",
  "@odata.type":
"#ManagerNetworkProtocol.v1_4_0.ManagerNetworkProtocol",
  "Description": "Manager Network
Service",
  "FQDN": "palmetto",
  "HTTP": {
    "Port": 0,
    "ProtocolEnabled": false
  },
  "HTTPS": {
    "Certificates": {
      "@odata.id": "/redfish/v1/Managers/bmc/Ne
tworkProtocol/HTTPS/Certificates"
    },
    "Port": 443,
    "ProtocolEnabled": true
  },
  "HostName": "palmetto",
  "IPMI": {
    "Port": 623,
    "ProtocolEnabled": true
  },
  "Id": "NetworkProtocol",
  "NTP": {
    "NTPServers": [
      "192.168.0.99"
    ],
    "ProtocolEnabled": true
  },
  "Name": "Manager Network
Protocol",
  "SSH": {
    "Port": 22
```

As it is always good to have the correct date and time in a BMC, you may want to supply at least one server IP in the `NTPServers` array of your BMC. To reach that goal you have first to verify in the Redfish Schema whether the `NTPServers` array can be modified.

Generally speaking the location of the Redfish Schemas of a particular OData type is under the `/redfish/v1/JsonSchemas/{type}` endpoint.

Run the following command listing the location(s) of the `ManagerNetworkProtocol` schema used by your BMC and study its output.

The **`PublicationUri`** URI requires an Internet connection to reach `http://redfish.dmtf.org`.

However, the `URI` pointer does not require any Internet access to view its content as it is embedded in the BMC at `/redfish/v1/JsonSchemas/ManagerNetworkProtocol/ManagerNetworkProtocol.json`

[9]:
```
echo "Manager Network Protocol schema locations:"

curl --insecure --noproxy "localhost, 127.0.0.1" --silent \
     --header "$HeaderContentType" --header "$HeaderODataVersion" \
     --header "X-Auth-Token: $BmcToken" \
     --request GET \
     ${BmcURI}/redfish/v1/JsonSchemas/ManagerNetworkProtocol | jq
```

```
Manager Network Protocol schema locations:
```

```
{
  "@odata.context":
"/redfish/v1/$metadata#JsonSchemaFile.JsonSchemaFile",
  "@odata.id":
"/redfish/v1/JsonSchemas/ManagerNetworkProtocol",
  "@odata.type":
"#JsonSchemaFile.v1_0_2.JsonSchemaFile",
  "Name": "ManagerNetworkProtocol Schema
File",
  "Schema":
"#ManagerNetworkProtocol.ManagerNetworkProtocol",
  "Description": "ManagerNetworkProtocol Schema
File Location",
  "Id": "ManagerNetworkProtocol",
  "Languages": [
    "en"
  ],
  "Languages@odata.count": 1,
  "Location": [
    {
      "Language": "en",
      "PublicationUri": "http://redfish.dmtf.or
g/schemas/v1/ManagerNetworkProtocol.json",
      "Uri": "/redfish/v1/JsonSchemas/ManagerNe
tworkProtocol/ManagerNetworkProtocol.json"
    }
  ],
  "Location@odata.count": 1
}
```

Using the embedded `URI` You can retrieve the definition of the `NTPServers` object and verify you will be able to modify it.

Run the following `curl` command which extracts the `NTPServers` schema definition and note the **`readonly = false`** property.

[10]:
```
echo "NTPServers schema definition:"

curl --insecure --noproxy "localhost, 127.0.0.1" --silent \
     --header "$HeaderContentType" --header "$HeaderODataVersion" \
     --header "X-Auth-Token: $BmcToken" \
     --request GET \
     ${BmcURI}/redfish/v1/JsonSchemas/ManagerNetworkProtocol/
 ↪ManagerNetworkProtocol.json | \
     jq '.definitions | .NTPProtocol | .properties | .NTPServers'
```

```
NTPServers schema definition:
{
  "description": "Indicates to which NTP
servers this manager is subscribed.",
  "items": {
    "type": [
      "string",
      "null"
    ]
  },
  "longDescription": "This property shall
contain all the NTP servers for which this manager is using to obtain
time.",
  "readonly": false,
  "type": "array",
  "versionAdded": "v1_2_0"
}
```

You are now sure that it is possible to alter/populate the list of `NTPServers` in your BMC.

The following commands performs a `PATCH` of the `NetworkProtocol` endpoint with a single NTP server IP address.

This `PATCH` request does not return any response data. Other Redfish implementation (i.e. HPE iLO) are more verbose. However, by checking the response header file `$ResponseHeaders`, you should see an `HTTP/1.1: 204` return code stating that the request was successful.

[11]:
```
echo "Patching NTP Servers"
curl --dump-header  $ResponseHeaders \
     --insecure --noproxy "localhost, 127.0.0.1" --silent \
     --header "$HeaderContentType" --header "$HeaderODataVersion" \
     --header "X-Auth-Token: $BmcToken" \
     --request PATCH --data '{ "NTP": {"NTPServers": ["192.168.0.99", ""]} }' \
     ${BmcURI}/redfish/v1/Managers/bmc/NetworkProtocol | jq
```

```
Patching NTP Servers
```

Verify with the following command that the `NTPServers` list contains the IP address you supplied.

[12]:
```
echo "NTP Server list:"
curl --insecure --silent --noproxy "localhost, 127.0.0.1"  \
  --header  "$HeaderContentType" --header "$HeaderODataVersion" \
  --header "X-Auth-Token: $BmcToken" \
  --request GET ${BmcURI}/redfish/v1/Managers/${bmc}/NetworkProtocol | jq '.NTP'
```

```
NTP Server list:
{

  "NTPServers": [

    "192.168.0.99",

    ""

  ],

  "ProtocolEnabled": true

}
```

1.7 Perform an action: Reset OpenBMC

In the previous section you modified a resource that is not requiring a reset of the BMC to be taken into account. However other parameters may require a restart when changed.

In this paragraph you will perform the `GracefulRestart` action seen previously in your OpenBMC using a `POST` request toward the corresponding target.

After you run this reset command, run the next `bash` cell in order to wait until the BMC is back online.

[13]:
```
echo "Starting a reset of the BMC at" ; date
echo

curl --insecure --noproxy "localhost, 127.0.0.1" --silent \
     --header "$HeaderContentType" --header "$HeaderODataVersion" \
     --header "X-Auth-Token: $BmcToken" \
     --request POST --data '{ "ResetType": "GracefulRestart"}' \
     ${BmcURI}/redfish/v1/Managers/bmc/Actions/Manager.Reset | jq
```

```
Starting a reset of the BMC at
Fri May 15 22:37:06 CEST 2020
```

```
{
  "@Message.ExtendedInfo": [
    {
      "@odata.type":
"#Message.v1_0_0.Message",
      "Message": "Successfully Completed
Request",
      "MessageArgs": [],
      "MessageId":
"Base.1.4.0.Success",
      "Resolution": "None",
      "Severity": "OK"
    }
  ]
}
```

1.8 Wait until OpenBMC is back online

The following cell loops until the BMC returns a valid output to a `GET` request. Run it and wait until the BMC is back on line. This should take about two minutes.

[14]:
```
ret=""
while [ "X${ret}" != "X0" ] ; do
    timeout 3 curl --insecure --noproxy "localhost, 127.0.0.1"  --silent \
      --header  "$HeaderContentType" --header "$HeaderODataVersion" \
      --header "X-Auth-Token: $BmcToken" \
      --request GET ${BmcURI}/redfish/v1/Managers/$bmc > /dev/null
    ret=$?
done

echo "BMC is back online at " ; date
echo
```

```
BMC is back online at
Fri May 15 22:39:12 CEST 2020
```

1.9 Delete sessions

It is extremely important to delete Redfish sessions to avoid reaching the maximum number of opened sessions in a BMC, preventing any access to it. Read this article for more detail.

[15]:
```
echo "Body response of a session deletion:"

curl --insecure --noproxy "localhost, 127.0.0.1" --silent \
     --header "$HeaderContentType" --header "$HeaderODataVersion" --header␣
 ↪"X-Auth-Token: $BmcToken" \
     --request DELETE $BmcSessionLocation | jq
```

```
Body response of a session deletion:
{

  "@odata.id":

"/redfish/v1/SessionService/Sessions/C8IJh1sHpg",

  "@odata.type":

"#Session.v1_0_2.Session",

  "Description": "Manager User

Session",

  "Id": "C8IJh1sHpg",

  "Name": "User Session",

  "UserName": "student"

}
```

1.9.1 What next ?

If you want to re-run this notebook against an **HPE iLO 5**, from your Jupyter Home page, you can duplicate it and then modify the `BmcIP` variable with `16.31.87.207`.

Then, you will be able to compare the output of OpenBMC and HPE iLO 5 Redfish implementations.

It is time now to go through the **Lab 3 notebook** to study a Python code suitable for several Redfish implementation