1-RedfishBash

November 30, 2020

Powered by HPE DEV Team

0.0.1 Redfish REST API overview

Version 0.76

0.1 What is Redfish?

As per the Redfish home page, DMTF Redfish® is a standard designed to deliver **simple and secure management** for converged, hybrid IT and the Software Defined Data Center (SDDC)".

0.2 Setting the scene

This Jupyter Notebook defines environment variables that will be used throughout the rest of the notebook. Using these variables, you will explore the Redfish resource tree and learn the session authentication mechanism using Bash and the cURL tool against an OpenBMC simulator. Then, you will modify a property and perform a reset of the OpenBMC before logging out.

For didactic reasons, commands presented in this notebook are not optimized.

The following `bash` code defines environment variables (i.e. IP address, username, password) associated with your student ID number that is stored in variable `$Stud`. It also creates a `.json` file that contains the credentials for your OpenBMC appliance that are 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]:
```
########## Environment preparation ((Version: 0.1)) #############

# Set Student ID number
Stud=00
```

```
# location and ports variables
let OpenBmcPort=44000+${Stud}
let iLO5SimulatorPort=45000+${Stud}
let ilo5Port=443


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

OpenBmcIP=bmcsimulators
OpenBmc="${OpenBmcIP}:${OpenBmcPort}"
OpenBmcURI="https://${OpenBmc}"


ilo5IP="ilo5"
#ilo5="${ilo5IP}:${ilo5Port}"
ilo5="${ilo5IP}"
ilo5URI="https://${ilo5}"

# Credentials
User="student"
Password='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 > ${SessionData} << __EOF__
{
        "UserName": "$User",
        "Password": "$Password"
}
__EOF__


# Verify we can reach the remote Bmcs on the right HTTPS ports.
for bmc in ilo5 OpenBmc 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
```

```
ilo5 is reachable
OpenBmc is reachable
iLO5Simulator is reachable
```

0.3 Retrieve the Redfish Service Entry point content (Root)

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

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

This request does not require any authentication.

If you are not familiar with cURL, refer to the manual page for help.

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

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

```
Available Redfish version(s):
{
  "v1": "/redfish/v1/"
}
```

The previous command returns only one available Redfish version for 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`, etc.
- 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 output.

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 mentioned 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 schema versions. Later, resources can be added, moved, or removed from one schema version to another.

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

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

0.3.1 Resource map (highlights)

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

- `AccountService`: a collection of user accounts present in this BMC.
- `Chassis`: a collection of chassis; physical view of the system containing global physical asset info (i.e. power, thermal). A system can have multiple chassis (e.g. HPE Superdome Flex).
- `Sessions`: a collection of current open sessions (ssh, https, GUI…).
- `Managers`: a collection of BMCs. A server can have multiple BMCs (i.e. HPE Moonshot have one iLO per SOC).
- `Systems`: a 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 Redfish System).

0.4 Create a Redfish session using `cURL`

All the URIs below the Root entry point require authentication. In this section, you'll go through the session authentication method for Redfish 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" \
     ${OpenBmcURI}/redfish/v1/SessionService/Sessions | jq
```

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

```
{
  "@odata.id":
"/redfish/v1/SessionService/Sessions/wTp9kNG9rE",
  "@odata.type":
"#Session.v1_0_2.Session",
  "Description": "Manager User
Session",
  "Id": "wTp9kNG9rE",
  "Name": "User Session",
  "UserName": "student"
}
```


0.5 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 when authentication is required. The `Session Location` will be used to close the session.

Play the next cell.

[5]:
```
BmcToken=$(awk '/X-Auth-Token/ {print $NF}' $ResponseHeaders | tr -d '\r')
BmcSessionLocation="$OpenBmcURI"$(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: ffmuk9D7pe913QwkvoAD
Bmc Session Location:
https://bmcsimulators:44000/redfish/v1/SessionService/Sessions/wTp9kNG9rE
```

0.6 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 on 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 the next cell and note that there is only one BMC present in your OpenBMC appliance (`Member@odata.count = 1`). Also note 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 ${OpenBmcURI}/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. Then, for each BMC, it extracts its properties.

Run the next cell 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 in the lab.

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 value added 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 ${OpenBmcURI}/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 ${OpenBmcURI}/redfish/v1/Managers/${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":
"2020-11-30T13:59:32+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":
```

0.7 View and modify a property

If you want to view a specific property (i.e. network protocols) supported by your BMC, you can retrieve it with a `GET` of the `Property` URI mentioned in the output of the above `GET` request.

Run the next `cURL`cell. Its output should show an empty array of `NTPServers` (if not, contact your instructor).

[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 ${OpenBmcURI}/redfish/v1/Managers/${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": [],
    "ProtocolEnabled": true
  },
  "Name": "Manager Network
Protocol",
  "SSH": {
    "Port": 22,
    "ProtocolEnabled": true
  }
```

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 to this, you must first verify whether the `NTPServers` array can be modified in the Redfish schema.

Due to time constraints, we will skip this verification and assume that this property is writable.

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 implementations (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.

[9]:
```
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", ""]} }' \
     ${OpenBmcURI}/redfish/v1/Managers/bmc/NetworkProtocol | jq
```

```
Patching NTP Servers
```

Using the following command, verify that the `NTPServers` list contains the IP address you supplied.

[10]:
```
echo "NTP Server list:"
curl --insecure --silent --noproxy "localhost, 127.0.0.1"  \
  --header  "$HeaderContentType" --header "$HeaderODataVersion" \
  --header "X-Auth-Token: $BmcToken" \
  --request GET ${OpenBmcURI}/redfish/v1/Managers/${bmc}/NetworkProtocol | jq '.
↪NTP'
```

```
NTP Server list:
{
  "NTPServers": [
    "192.168.0.99",
    ""
  ],
  "ProtocolEnabled": true
}
```

0.8 Perform an action: Reset OpenBMC

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

In this section, 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.

[11]:
```
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"}' \
     ${OpenBmcURI}/redfish/v1/Managers/bmc/Actions/Manager.Reset | jq
```

```
Starting a reset of the BMC at
Mon 30 Nov 2020 03:00:17 PM CET

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

0.9 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 three minutes.

[12]:
```
ret=""
while [ "X${ret}" != "XO" ] ; do
    timeout 3 curl --insecure --noproxy "localhost, 127.0.0.1"  --silent \
      --header  "$HeaderContentType" --header "$HeaderODataVersion" \
      --header "X-Auth-Token: $BmcToken" \
```

```
      --request GET ${OpenBmcURI}/redfish/v1/Managers/$bmc > /dev/null
    ret=$?
done

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

```
BMC is back online at
Mon 30 Nov 2020 03:01:56 PM CET
```

0.10 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.

[13]:
```
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/wTp9kNG9rE",
  "@odata.type":
"#Session.v1_0_2.Session",
  "Description": "Manager User
Session",
  "Id": "wTp9kNG9rE",
  "Name": "User Session",
  "UserName": "student"
}
```

0.10.1 What next ?

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