2-RedfishPowerShell

November 30, 2020

Powered by HPE DEV Team

0.0.1 Redfish API Overview

Version 0.66

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 Microsoft PowerShell and its native invoke-WebRequest cmdlet 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 cell defines global 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]:
```
# Set Student ID number
$Stud=00
echo "You are Student $Stud"
```

```
# OpenBMC Host
$OpenBmcPort=44000 + $Stud
$BmcIP = "openbmcs:$OpenBmcPort"

# OpenBMC credentials
$user = "student"
$pass = "P@ssw0rd!"

# Convert the credentials to a base 64 encoded http Basic Auth
$pair = "${user}:${pass}"
$mybytes = [System.Text.Encoding]::ASCII.GetBytes($pair)
$b64 = [System.Convert]::ToBase64String($mybytes)
$basicAuthValue = "Basic $b64"
$AuthHeaders = @{ Authorization = $basicAuthValue }
```

```
You are Student 0
```

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) that are available today in your OpenBMC simulator.

This request does not require any authentication.

[2]:
```
$BmcURI = "https://${BmcIP}/redfish"
echo "Attempting HTTP GET @ $BmcURI"
# HTTP GET
$r = Invoke-WebRequest -SkipCertificateCheck -Uri $BmcURI -Method 'GET'␣
 ↪-ErrorAction Stop

# Format the JSON Body Response
ConvertFrom-Json $r.Content
```

```
Attempting HTTP GET @ https://openbmcs:44000/redfish

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 PowerShell Invoke-WebRequest cmdlet 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]:
```
$BmcURI = "https://${BmcIP}/redfish/v1"

echo "Attempting HTTP GET @ $BmcURI"
# HTTP GET
$r = Invoke-WebRequest -SkipCertificateCheck -Uri $BmcURI -Method 'GET'␣
 ↪-ErrorAction Stop

# Format the JSON Body Response
ConvertFrom-Json $r.Content
```

```
Attempting HTTP GET @ https://openbmcs:44000/redfish/v1

@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=}
Managers           : @{@odata.id=/redfish/v1/Managers}
Name               : Root Service
RedfishVersion     : 1.9.0
Registries         : @{@odata.id=/redfish/v1/Registries}
SessionService     : @{@odata.id=/redfish/v1/SessionService}
Systems            : @{@odata.id=/redfish/v1/Systems}
Tasks              : @{@odata.id=/redfish/v1/TaskService}
UUID               : 8a557334-4a18-4b2a-9b80-da497882408c
UpdateService      : @{@odata.id=/redfish/v1/UpdateService}
```

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 PowerShell

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 PowerShell `Invoke-WebRequest` cmdlet sends a `POST` request toward the standard `/redfish/v1/SessionService/Sessions` URI of your BMC. The body/workload of this request is in the `@${Body}` json data populated using the credentials from the very first `PowerShell` cell of this notebook (Environment variables).

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 `$r_headers` variable and further parsed into the $token and $location variables.

Play the next cell.

[4]:
```
# Create the JSON request to pass BMC credentials to the API
$Body = @{
    UserName = $user
    Password = $pass
}
$Body = ($Body|ConvertTo-Json)

$BmcURI = "https://${BmcIP}/redfish/v1/SessionService/Sessions"

$r = Invoke-WebRequest -SkipCertificateCheck -Uri $BmcURI -Method 'POST' -Body
$Body -ErrorAction Stop

$r_headers = $r.Headers
$token = $r_headers."X-Auth-Token"[0]
$location = $r_headers."Location"[0]

echo "Bmc Token: $token"
```

```
echo "Bmc Session Location $location"
```

```
Bmc Token: E1Lgnr5gC1GY6IdXKAYW
Bmc Session Location /redfish/v1/SessionService/Sessions/Q6KhGQKFzG
```

0.5 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 lab infrastructure is based on servers with only a single BMC. However, we'll use 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 **it requires authentication**, you must supply the **`X-Auth-Token`** as part of the header.

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`.

[5]:
```
$headers = @{'X-Auth-Token'= $token}

$BmcURI = "https://${BmcIP}/redfish/v1/Managers"
echo "Attempting HTTP GET @ $BmcURI with X-Auth-Token: $token"

$r = Invoke-WebRequest -SkipCertificateCheck -Uri $BmcURI -Method 'GET'␣
 ↪-Headers $headers -ErrorAction Stop

ConvertFrom-Json $r.Content
```

```
Attempting HTTP GET @ https://openbmcs:44000/redfish/v1/Managers with X-Auth-
Token: E1Lgnr5gC1GY6IdXKAYW

@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.

[6]:
```
$managers = ConvertFrom-Json $r.Content
$bmcs = $()
$bmcuris = $()

foreach ($member in $managers.Members) {
  $url = $member.'@odata.id'
  $bmcuris += "https://${BmcIP}${url}"
  $bmcs += $url.substring($url.lastindexof("/")+1)
}

echo "List of BMC(s) present in this system: " $bmcs

foreach ($uri in $bmcuris) {
  echo "Properties of BMC: " $uri.substring($uri.lastindexof("/")+1)
  echo ""
  $bmc_r = Invoke-WebRequest -SkipCertificateCheck -Uri $uri -Method 'GET'␣
 ↪-Headers $headers -ErrorAction Stop
  $bmc_r.Content
}
```

```
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": "2020-11-30T14:11:59+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": "/redfish/v1/Chassis/chassis"
    }
  ],
  "ManagerForChassis@odata.count": 1,
  "ManagerForServers": [
    {
      "@odata.id": "/redfish/v1/Systems/system"
    }
  ],
  "ManagerForServers@odata.count": 1,
  "ManagerInChassis": {
    "@odata.id": "/redfish/v1/Chassis/chassis"
  }
},
"LogServices": {
  "@odata.id": "/redfish/v1/Managers/bmc/LogServices"
},
"ManagerType": "BMC",
"Model": "OpenBmc",
"Name": "OpenBmc Manager",
"NetworkProtocol": {
  "@odata.id": "/redfish/v1/Managers/bmc/NetworkProtocol"
},
"Oem": {
  "@odata.id": "/redfish/v1/Managers/bmc#/Oem",
  "@odata.type": "#OemManager.Oem",
  "OpenBmc": {
    "@odata.id": "/redfish/v1/Managers/bmc#/Oem/OpenBmc",
    "@odata.type": "#OemManager.OpenBmc",
    "Certificates": {
      "@odata.id": "/redfish/v1/Managers/bmc/Truststore/Certificates"
    }
  }
},
"PowerState": "On",
```

```
  "SerialConsole": {
    "ConnectTypesSupported": [
      "IPMI",
      "SSH"
    ],
    "MaxConcurrentSessions": 15,
    "ServiceEnabled": true
  },
  "ServiceEntryPointUUID": "8a557334-4a18-4b2a-9b80-da497882408c",
  "Status": {
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  },
  "UUID": "437d2061-659b-4157-9afa-95bf26c8f42d"
}
```


0.6 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 `PowerShell`cell. Its output should show an empty array of `NTPServers` (if not, contact your instructor).

[7]:
```
$BmcURI = "https://${BmcIP}/redfish/v1/Managers/bmc/NetworkProtocol"

$net_r = Invoke-WebRequest -SkipCertificateCheck -Uri $BmcURI -Method 'GET'␣
 ↪-Headers $headers -ErrorAction Stop

$net_r.Content
```


```
{
  "@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/NetworkProtocol/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
  },
  "Status": {
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  }
}
```

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.

[8]:
```
$BmcURI = "https://${BmcIP}/redfish/v1/Managers/bmc/NetworkProtocol"
$Body = @{
  NTP = @{
    NTPServers = "192.168.0.99", ""
  }
}
$Body = ($Body|ConvertTo-Json)

$ignore = Invoke-WebRequest -SkipCertificateCheck -Uri $BmcURI -Method 'PATCH'␣
 ↪-Body $Body -Headers $headers -ErrorAction Stop
```

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

[9]:
```
$BmcURI = "https://${BmcIP}/redfish/v1/Managers/bmc/NetworkProtocol"

$patch_results = Invoke-WebRequest -SkipCertificateCheck -Uri $BmcURI -Method␣
 ↪'GET' -Headers $headers -ErrorAction Stop

$ntp = $patch_results.Content | ConvertFrom-Json
$ntp[0].NTP
```

```
NTPServers     ProtocolEnabled
----------     ---------------
{192.168.0.99}            True
```

0.7 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 `powershell` cell in order to wait until the BMC is back online.

[10]:
```
$BmcURI = "https://${BmcIP}/redfish/v1/Managers/bmc/Actions/Manager.Reset"
$Body = @{
  "ResetType" = "GracefulRestart"
}
$Body = ($Body|ConvertTo-Json)

$date = Get-Date
echo "Starting a reset of the BMC at $date"

$ret = Invoke-WebRequest -SkipCertificateCheck -Uri $BmcURI -Method 'POST'␣
 ↪-Body $Body -Headers $headers -ErrorAction Stop

$ret.Content
```

```
Starting a reset of the BMC at 11/30/2020 14:13:52
{
  "@Message.ExtendedInfo": [
    {
      "@odata.type": "#Message.v1_0_0.Message",
      "Message": "Successfully Completed Request",
      "MessageArgs": [],
      "MessageId": "Base.1.4.0.Success",
      "Resolution": "None",
      "Severity": "OK"
```

```
    }
  ]
}
```

0.8 Wait until OpenBMC is back online

The following cell loops until the BMC returns a valid HTTP 200 response to a `GET` request. Run it and wait until the BMC is back on line. This should take about three minutes.

[11]:
```
$BmcURI = "https://${BmcIP}/redfish/v1/Managers/bmc"

Do {
  try {
    $r = Invoke-WebRequest -SkipCertificateCheck -Uri $BmcURI -Method 'GET'␣
 ↪-Headers $headers -TimeoutSec 1 -ErrorAction Stop
    $StatusCode = $r.StatusCode
  }
  catch
  {
    $StatusCode = $_.Exception.Response.StatusCode.value__
    Start-Sleep -s 1
  }

} Until ($StatusCode -eq 200)
$date = Get-Date
echo "BMC is back online at $date"
```

```
BMC is back online at 11/30/2020 14:15:35
```

0.9 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.

[12]:
```
$BmcURI = "https://${BmcIP}$location"

$r = Invoke-WebRequest -SkipCertificateCheck -Uri $BmcURI -Method 'DELETE'␣
 ↪-Headers $headers -ErrorAction Stop

$r.Content
```

```
{
  "@odata.id": "/redfish/v1/SessionService/Sessions/Q6KhGQKFzG",
  "@odata.type": "#Session.v1_0_2.Session",
  "Description": "Manager User Session",
  "Id": "Q6KhGQKFzG",
  "Name": "User Session",
  "UserName": "student"
}
```

0.9.1 What next ?

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