2-Aspire-RedfishPowerShell

May 15, 2020

1 Hack Shack

Powered by HPE DEV Team

1.0.1 Speakers : Troy Heber, François Donzé

1.0.2 Redfish API Overview workshop

Abstract: Redfish, a DMTF management standard is now recognized as the best solution to replace proprietary protocols. During this follow along Hack Shack workshop, you will interact with this HTTP/JSON based standard using your favorite language: PowerShell, Python or Bash/cURL. You will explore the Redfish tree of an OpenBMC and an HPE iLO 5 to understand its basic structure. In addition you will learn how to modify resources and perform simple actions. Beginners and experts are welcome.

Version 0.51

1.1 Introduction

This notebook contains a single PowerShell program, explained and executed step by step. Its goal is to retrieve the BMC MAC addresses from an OpenBMC and from an HPE iLO 5 using a single piece of code.

The account on your dedicated OpenBMC simulator has Administrator privileges while you have Read-Only persmissions on the physical shared iLO 5.

For simplicity and didactic reasons, the following code is not optimized and does not handle errors or test return codes. The simple syntax used should be easy to understand by non-PowerShell knowledgeable students.

1.1.1 Setting the scene

The following cell imports the minimum required PowerShell Cmdlets and defines **global variables** (credentials and IP addresses).

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

# OpenBMC Host
$BmcIP = "16.31.86.70:443${Stud}"
$BmcURI = "https://${BmcIP}/redfish/"

# 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 00
```

1.2 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.

[2]:
```
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://16.31.86.70:44300/redfish/
```

```
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 PowerShell Invoke-WebRequest cmdlet 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]:
```
$BmcURI = "https://${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://16.31.86.70:44300/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}
```

1.2.1 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).

1.3 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 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.

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

```
#echo "Body: " ($Body|ConvertTo-Json)                                        ␣
 ↪
$BmcURI = "https://${BmcIP}/redfish/v1/SessionService/Sessions/"

#echo "Attempting HTTP PUT @ $BmcURI to authenticate to establish a session and␣
 ↪get an X-Auth-Token"
$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: dBrlKSQMfANu26JzaHf3
Bmc Session Location /redfish/v1/SessionService/Sessions/v3VjpbCTVU
```

1.4 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 upon 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 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`.

[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://16.31.86.70:44300/redfish/v1/Managers with X-Auth-
Token: dBrlKSQMfANu26JzaHf3

@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.

[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": "2020-05-15T20:43:29+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"
}
```

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 `PowerShell`cell. 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.

[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 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

```
[8]: $BmcURI = "https://${BmcIP}/redfish/v1/JsonSchemas/ManagerNetworkProtocol"

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

     $sch_r.Content
```

```
{
    "@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.org/schemas/v1/ManagerNetworkProtocol.json",
            "Uri":
"/redfish/v1/JsonSchemas/ManagerNetworkProtocol/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 to extract the NTPServers schema definition and note the **readonly = false** property.

```
[9]: $BmcURI = "https://${BmcIP}/redfish/v1/JsonSchemas/ManagerNetworkProtocol/
      ↪ManagerNetworkProtocol.json"

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

     $content = $def_r.Content | ConvertFrom-Json
```

```
$content[0].definitions.NTPProtocol |ConvertTo-Json
```

```
{
  "additionalProperties": false,
  "description": "The settings for a network protocol associated with a
manager.",
  "longDescription": "This type shall describe information about a protocol
setting for a manager.",
  "patternProperties": {
    "^([a-zA-Z_][a-zA-Z0-9_]*)?@(odata|Redfish|Message)\\.[a-zA-
Z_][a-zA-Z0-9_]*$": {
      "description": "This property shall specify a valid odata or Redfish
property.",
      "type": "array boolean integer number null object string"
    }
  },
  "properties": {
    "NTPServers": {
      "description": "Indicates to which NTP servers this manager is
subscribed.",
      "items": "@{type=System.Object[]}",
      "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"
    },
    "Port": {
      "description": "The protocol port.",
      "longDescription": "This property shall contain the port assigned to the
protocol.",
      "minimum": 0,
      "readonly": false,
      "type": "integer null"
    },
    "ProtocolEnabled": {
      "description": "An indication of whether the protocol is enabled.",
      "longDescription": "This property shall indicate whether the protocol is
enabled.",
      "readonly": false,
      "type": "boolean null"
    }
  },
  "type": "object"
}
```

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.

[10]:
```
$BmcURI = "https://${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
```

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

[11]:
```
$BmcURI = "https://${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
```

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

[12]:
```
$BmcURI = "https://${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 05/15/2020 22:44:11
{
  "@Message.ExtendedInfo": [
    {
      "@odata.type": "#Message.v1_0_0.Message",
      "Message": "Successfully Completed Request",
      "MessageArgs": [],
      "MessageId": "Base.1.4.0.Success",
      "Resolution": "None",
      "Severity": "OK"
    }
  ]
}
```

1.6 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 two minutes.

[13]:
```
$BmcURI = "https://${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 05/15/2020 22:46:02
```

1.7 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.

[14]:
```
$BmcURI = "https://${BmcIP}$location"

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

$r.Content
```

```
{
  "@odata.id": "/redfish/v1/SessionService/Sessions/v3VjpbCTVU",
  "@odata.type": "#Session.v1_0_2.Session",
  "Description": "Manager User Session",
  "Id": "v3VjpbCTVU",
  "Name": "User Session",
  "UserName": "student"
}
```

1.7.1 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