Mastering Server Management with PowerShell and Redfish Protocol

How tf did they get these people to agree on a standard?!

Blake Cherry

X: @blakelishly

Frank Lesniak

X: @franklesniak

```
C:\Users\bcherry>whoami
Blake Cherry (@blakelishly)
Cybersecurity & Enterprise Technology Senior Consultant at West Monroe

Experience:
Four years of consulting on automation/infrastructure for divestitures and integrations

Credentials:
Leads greenfield infrastructure & M365 greenfield build automation at West Monroe

**************************************************************************************************************
Ask Me About:

- PowerShell and Infrastructure as Code Automation
- My home-lab and self-hosted services
- Azure Cloud
- DevOps and pipeline automation
- Purdue Boilermakers football and basketball

**************************************************************************************************************
Contact:

- x.com/blakelishly
- linkedin.com/in/blake-cherry-333053148
- github.com/blakelishly
- bcherry@westmonroe.com


C:\Users\bcherry>
```

Command Prompt

```
C:\Users\flesniak>whoami
Frank Lesniak (@franklesniak)
Sr. Cybersecurity & Enterprise Technology Architect at West Monroe

Experience:
Almost 20 years in consulting. Microsoft 365 (modern work) consulting team lead.

Credentials:
MCSEs. PowerShellin' since 2006.

***************************************************************************************************************
Ask Me About:

- PowerShell automation (duh!)
- Executing corporate divestitures and integrations - where I spent most of my consulting client time
- File server modernization - moving to Azure Files, Teams/SharePoint, etc.
- Retro video game software and FPGA emulation
- My upcoming home construction project

***************************************************************************************************************
Contact:

- x.com/franklesniak
- linkedin.com/in/flesniak
- github.com/franklesniak
- bsky.app/profile/franklesniak.com or infosec.exchange/@franklesniak
- flesniak ATSIGN westmonroe.com


C:\Users\flesniak>
```

Thank you to our sponsors!

System Frontier

TEKsystems®
Own change

IRONMAN SOFTWARE

Chocolatey

Agenda

- Introduction and Key Concepts
- Understanding Redfish: The Modern Standard for Data Center Management
- Using PowerShell with Redfish
- Preparing Your Environment: Getting Ready for Redfish Development
- Redfish PowerShell Demos:
  - Automating Data Center Inventory Collection
  - Centralized System Log Collection for Faster Troubleshooting
  - Executing Administrative Actions on Data Center Hardware

What is a Baseboard Management Controller (BMC)?

A **Baseboard Management Controller (BMC)** is a specialized microcontroller embedded on server motherboards, typically responsible for managing and monitoring the hardware.

Think of the BMC as a mini-computer, like a Raspberry Pi, designed to interface with and control the larger server. It's equipped with its own ARM-based SoC, memory, and network interfaces, which provide multiple capabilities, usually including:

- Remote Power Management
- Hardware Monitoring
- Firmware Management
- Console Access

What is a Baseboard Management Controller (BMC)?

A **Baseboard Management Controller (BMC)** is a specialized microcontroller embedded on server motherboards, typically responsible for managing and monitoring the hardware.

Think of the BMC as a mini-computer, like a Raspberry Pi, designed to interface with and control the larger server. It's equipped with its own ARM-based SoC, memory, and network interfaces, which provide multiple capabilities, usually including:

- Remote Power Management
- Hardware Monitoring
- Firmware Management
- Console Access

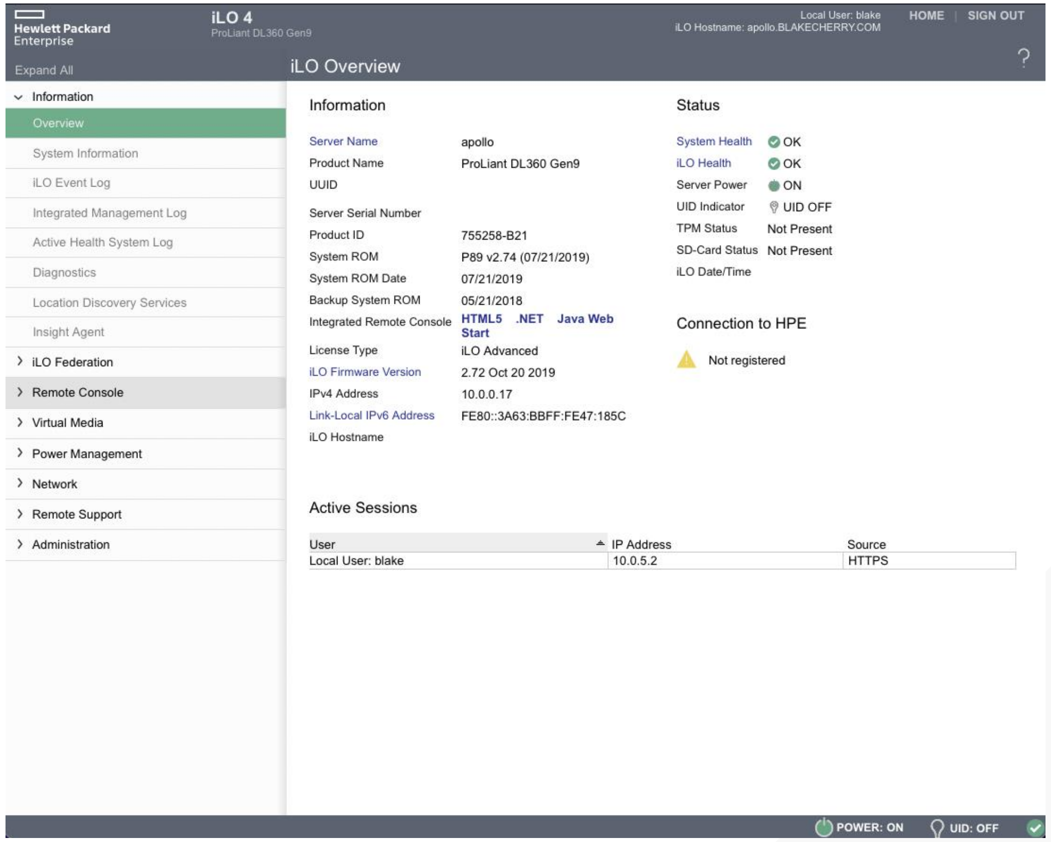
Most enterprise hardware manufacturers integrate BMCs into their modern products. Some popular implementations are:

- Dell iDRAC
- HP iLO
- Lenovo XClarity Controller

How do you interact with a BMC?

You can typically interact with most BMCs remotely in multiple ways; however, implementations vary across manufacturers. Historically, the primary methods are:

1. **Web Interface**
   - Most BMCs provide a built-in web interface for administrators.



Information

Server Name	apollo
Product Name	ProLiant DL360 Gen9
UUID	
Server Serial Number	
Product ID	755258-B21
System ROM	P89 v2.74 (07/21/2019)
System ROM Date	07/21/2019
Backup System ROM	05/21/2018
Integrated Remote Console	HTML5 .NET Java Web Start
License Type	iLO Advanced
iLO Firmware Version	2.72 Oct 20 2019
IPv4 Address	10.0.0.17
Link-Local IPv6 Address	FE80::3A63:BBFF:FE47:185C
iLO Hostname	

Status

System Health	OK
iLO Health	OK
Server Power	ON
UID Indicator	UID OFF
TPM Status	Not Present
SD-Card Status	Not Present
iLO Date/Time	

Connection to HPE

Not registered

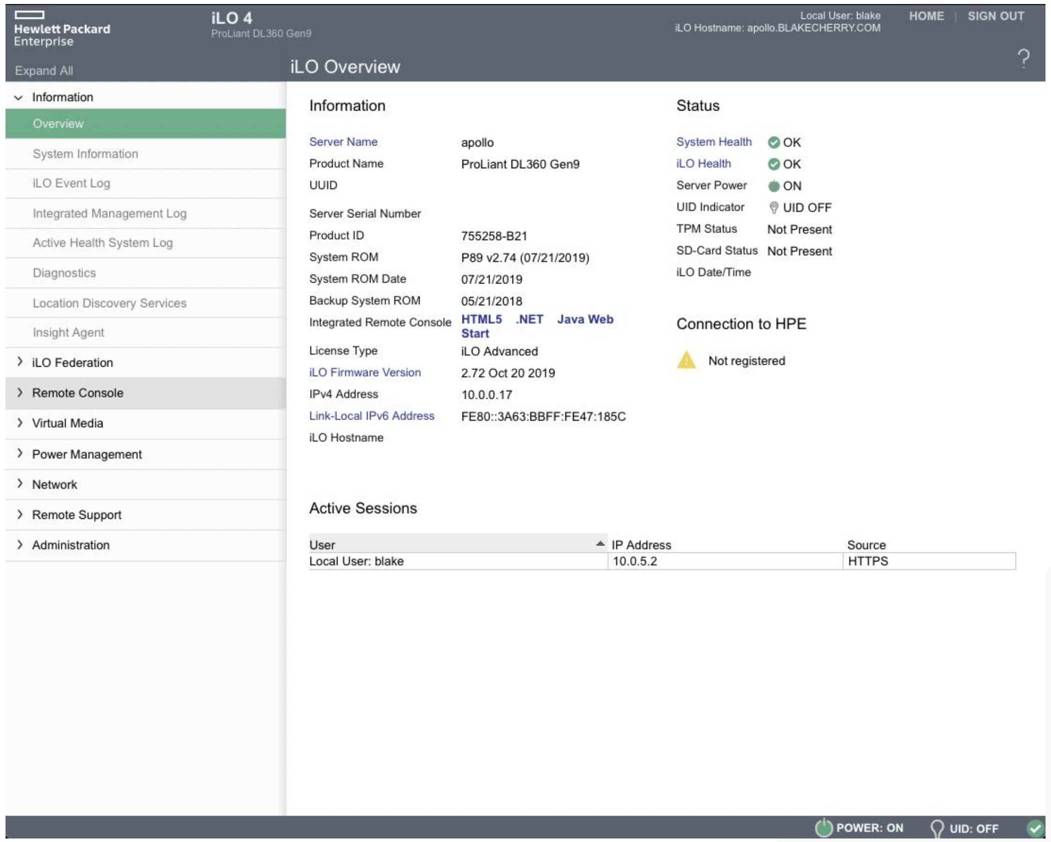
Active Sessions

User	IP Address	Source
Local User: blake	10.0.5.2	HTTPS

How do you interact with a BMC?

You can typically interact with most BMCs remotely in multiple ways; however, implementations vary across manufacturers. Historically, the primary methods are:

1. **Web Interface**
   - Most BMCs provide a built-in web interface for administrators.
2. **IPMI (Intelligent Platform Management Interface)**
   - IPMI has long been the standard for out-of-band management. It provides a command-line interface for administrators to interact with the BMC using a set of defined protocols.
   - IPMI was published in 1998 and sets standards for managing hardware components such as servers, but suffers from several key limitations:
     - **Poor scalability** and **lacks user-friendly features**, making it difficult to manage large, modern data centers
     - **Lacks modern security functions**, required by modern enterprise
     - **Interoperability issues**, leading to fragmented implementations that complicated cross-platform management



Information

Server Name	apollo
Product Name	ProLiant DL360 Gen9
UUID	
Server Serial Number	
Product ID	755258-B21
System ROM	P89 v2.74 (07/21/2019)
System ROM Date	07/21/2019
Backup System ROM	05/21/2018
Integrated Remote Console	HTML5 .NET Java Web Start
License Type	iLO Advanced
iLO Firmware Version	2.72 Oct 20 2019
IPv4 Address	10.0.0.17
Link-Local IPv6 Address	FE80::3A63:BBFF:FE47:185C
iLO Hostname	

Status

System Health	OK
iLO Health	OK
Server Power	ON
UID Indicator	UID OFF
TPM Status	Not Present
SD-Card Status	Not Present
iLO Date/Time	

Connection to HPE

Not registered

Active Sessions

User	IP Address	Source
Local User: blake	10.0.5.2	HTTPS

What is Redfish?

Developed by the Distributed Management Task Force (DMTF), Redfish is a network standard and API created to standardize and modernize the way administrators interact with BMCs.

Offering a RESTful API interface, it can easily be integrated with modern tools and automation platforms.

The first version of the Redfish standard, Redfish 1.0, was released in 2015, introducing base schema models that defined which resources Redfish could manage and how to interact with them. Subsequent releases of the specification have expanded these schemas to include additional system components and functionality.

What is Redfish?

Developed by the Distributed Management Task Force (DMTF), Redfish is a network standard and API created to standardize and modernize the way administrators interact with BMCs.

Offering a RESTful API interface, it can easily be integrated with modern tools and automation platforms.

The first version of the Redfish standard, Redfish 1.0, was released in 2015, introducing base schema models that defined which resources Redfish could manage and how to interact with them. Subsequent releases of the specification have expanded these schemas to include additional system components and functionality.

Redfish was designed to eventually wholly replace IPMI, and unlike its predecessor, Redfish is:

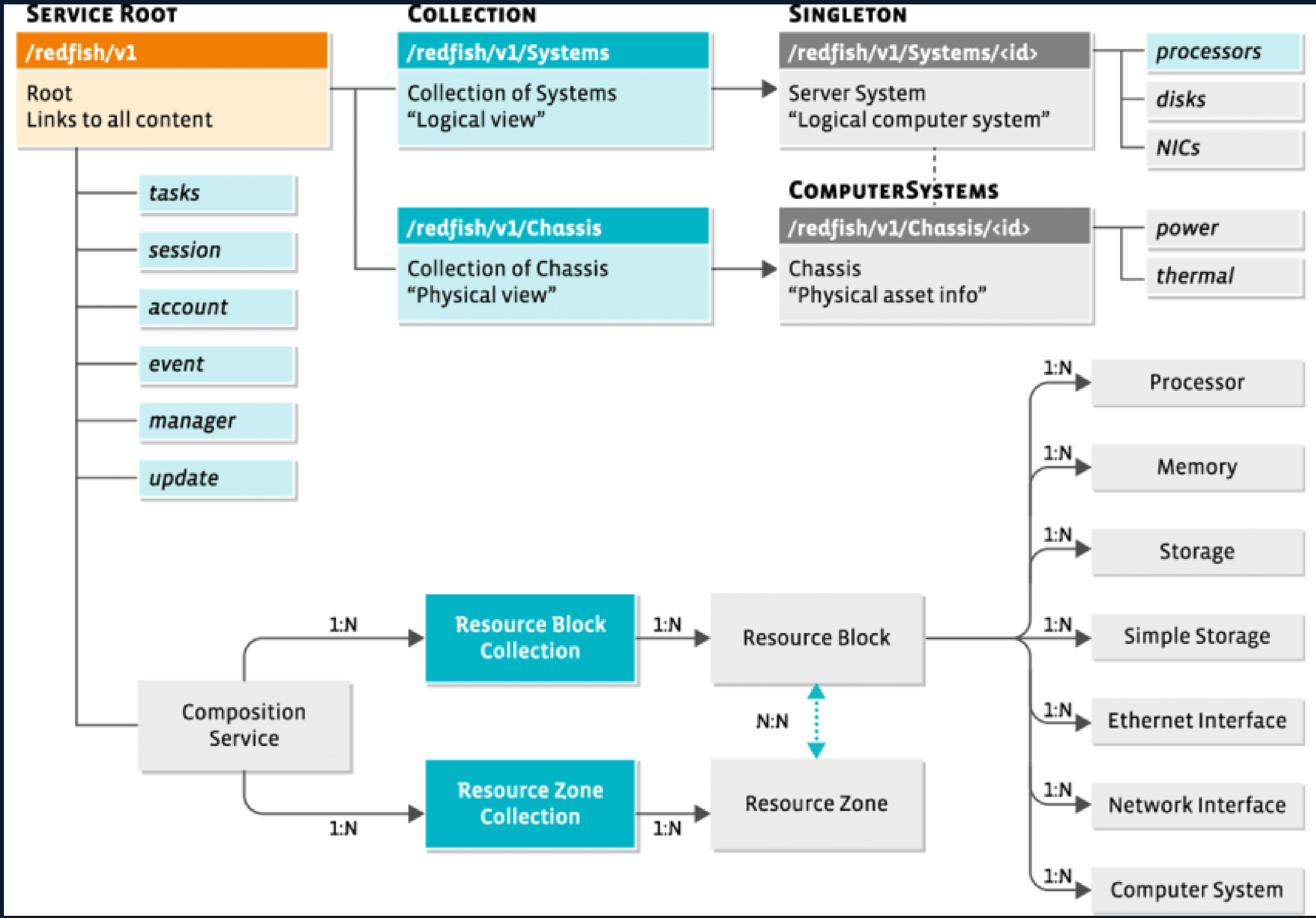
- **Web-Based and Human-Readable**: Uses JSON format, making it easier to understand and integrate into modern systems.
- **Extensible**: Unlike the fixed nature of IPMI, Redfish can be extended to handle new types of hardware as data center infrastructure evolves.
- **More Secure**: Supports TLS encryption, token-based authentication, and use of centralized identity providers.

Most Vendors Support Redfish

- ✓ DELL iDRAC BMC with Minimum iDRAC 7/8 FW 2.40.40.40, iDRAC9 FW 3.00.00.0
- ✓ HPE iLO BMC with minimum iLO4 FW 2.30, iLO5
- ✓ HPE Moonshot BMC with minimum FW 1.41
- ✓ Supermicro X10 BMC with minimum FW 3.0 and X11 with minimum FW 1.0
- ✓ Insyde Software Supervyse BMC
- ✓ Lenovo Clarity Controller XCC FW 1.00

THE REDFISH SCHEMA

- The Redfish data model is integrated into the standard, with components nested under the service root endpoint.
- While some manufacturers extend the schema, the core implementation remains consistent across all platforms.
- Objects are cross-referenced, allowing you to easily map physical components to their logical counterparts.
- Many components require authentication for access, ensuring secure data handling.

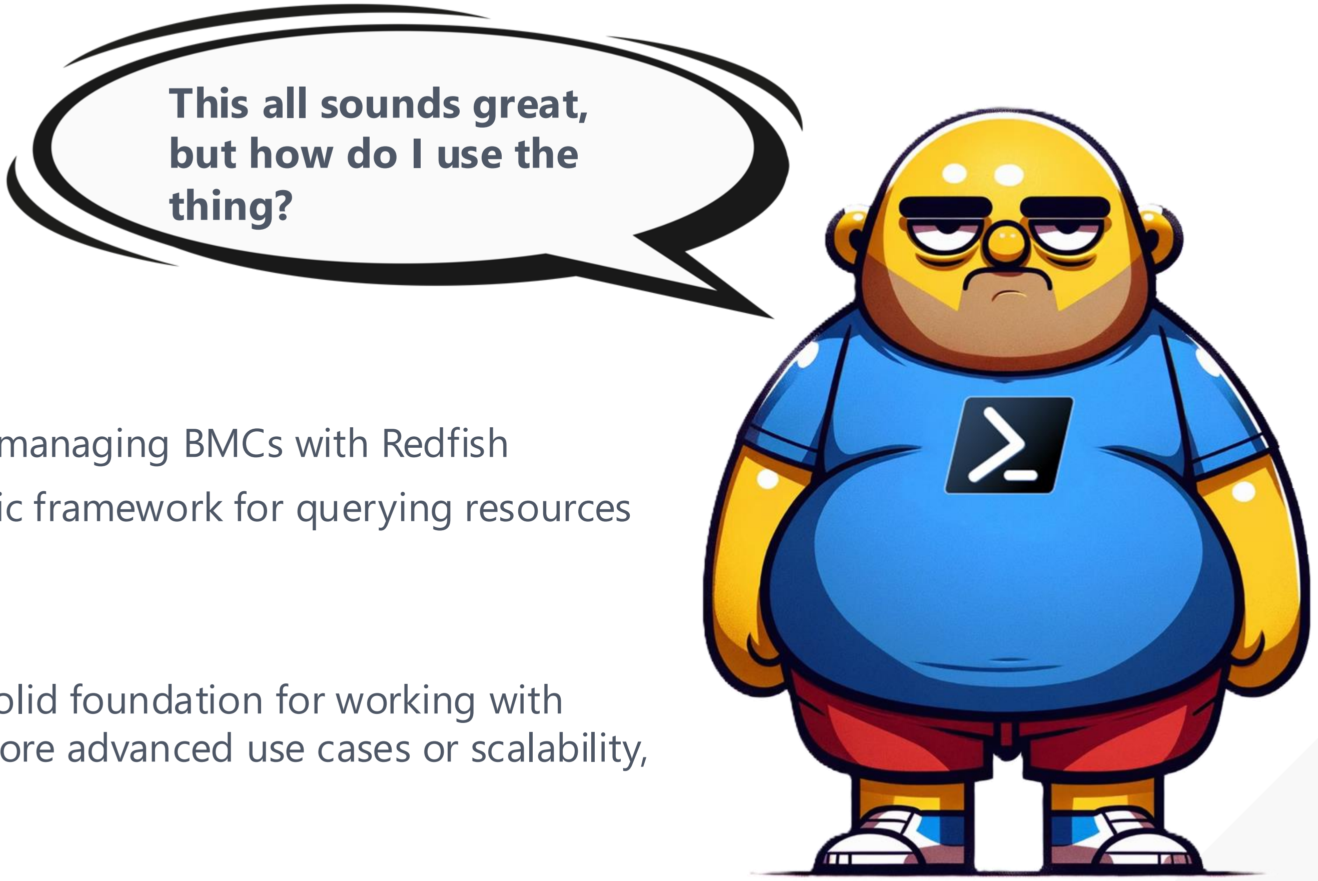


westMONROE

There are many open-source projects that provide tools and libraries to interact with Redfish.

There are many open-source projects that provide tools and libraries to interact with Redfish.

- RedfishTool – a Python34 program that implements a command line tool for accessing the Redfish API
- Gofish – a Golang library for Redfish
- Libredfish – a C client library for Redfish
- Python libraries:
  - Python Redfish Library
  - Python Redfish Utility
  - Sushy
- Ansible Redfish Module – an Ansible module for managing BMCs with Redfish
- PowerShell Toolkit for Swordfish – provides a basic framework for querying resources



While these open-source tools and libraries offer a solid foundation for working with Redfish, many provide only basic functionality. For more advanced use cases or scalability, custom scripts and code is typically required.

PowerShell is an effective tool for managing Redfish-enabled systems.

PowerShell is an ideal choice for interacting with Redfish-enabled systems.

With the ability to execute RESTful API calls and to easily manipulate data, PowerShell makes it easy to orchestrate automations or to streamline common tasks like hardware inventory, remote management, and system monitoring.

PowerShell is an effective tool for managing Redfish-enabled systems.

PowerShell is an ideal choice for interacting with Redfish-enabled systems.

With the ability to execute RESTful API calls and to easily manipulate data, PowerShell makes it easy to orchestrate automations or to streamline common tasks like hardware inventory, remote management, and system monitoring.

When authoring PowerShell scripts for Redfish, keep the following in mind:

- Changes to web cmdlets in PowerShell 7
  - The underlying .NET API of the Web Cmdlets has been changed to System.Net.Http.HttpClient
  - This introduces several breaking changes to Invoke-WebRequest and Invoke-RestMethod
- Platform-specific behavior
  - Interfacing with Redfish involves frequent manipulation of URL and directory paths. Keep in mind the differences in how each platform handles “/” and “\”
- Use functions and reusable code
  - Most functions supported by a Redfish endpoint require an authentication step. To reduce code duplication, using functions and custom object types is critical.

Preparing Your Environment: Getting Ready for Redfish Development

Understanding the API and Data Model:

- To determine which session-based operations (GET, PATCH, POST, DELETE) are supported by a specific Redfish implementation, consult the official Redfish standard or manufacturer documentation.
- Different manufacturers may extend the standard Redfish data model, so implementations can vary.
- The DMTF has published mockups of hardware standards' data models, which can be viewed interactively. HPE also offers an interactive API viewer with ProLiant mockups.
  - https://ilorestfulapiexplorer.ext.hpe.com/
  - https://redfish.dmtf.org/redfish/v1

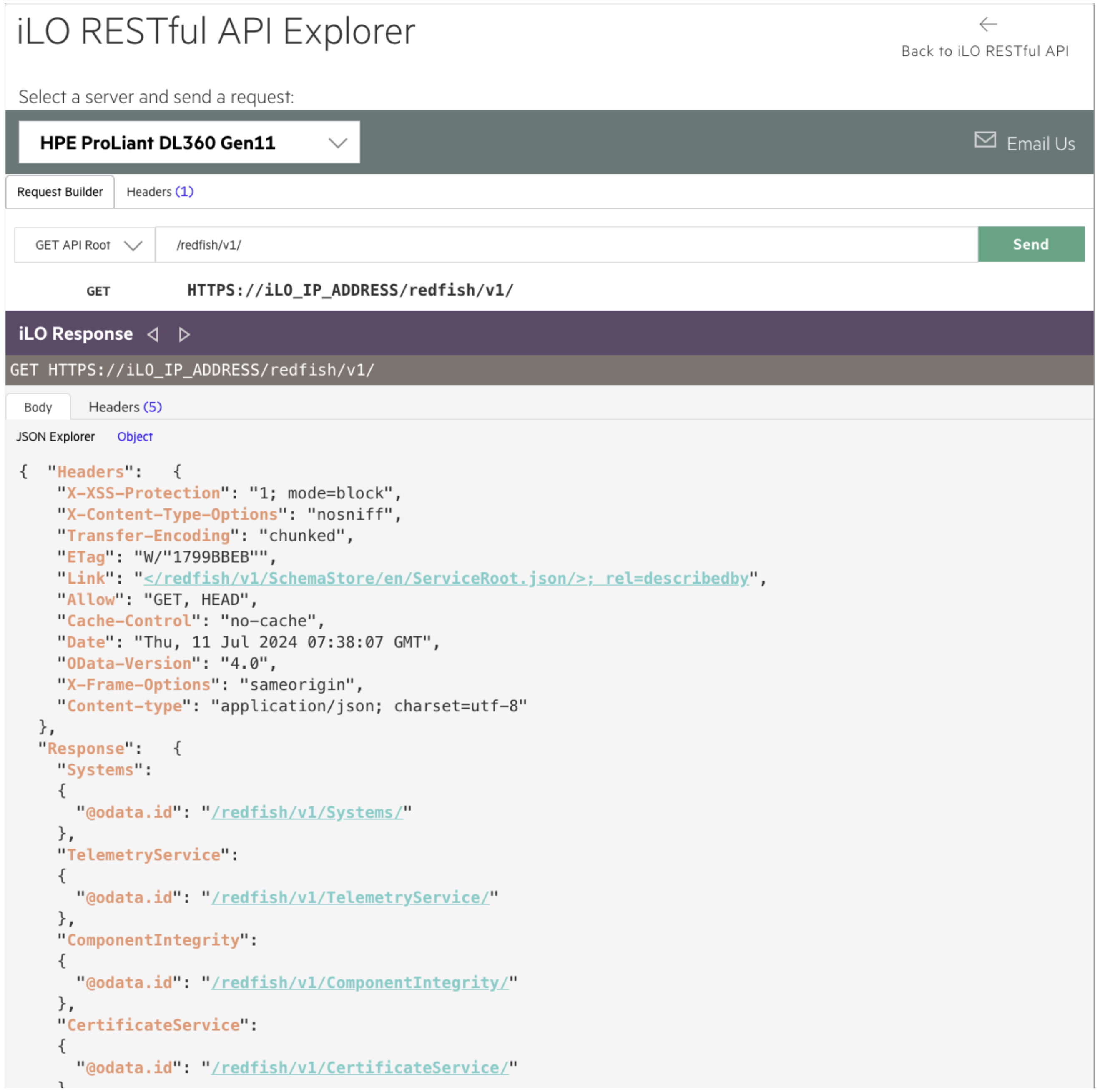
Preparing Your Environment: Getting Ready for Redfish Development

Understanding the API and Data Model:

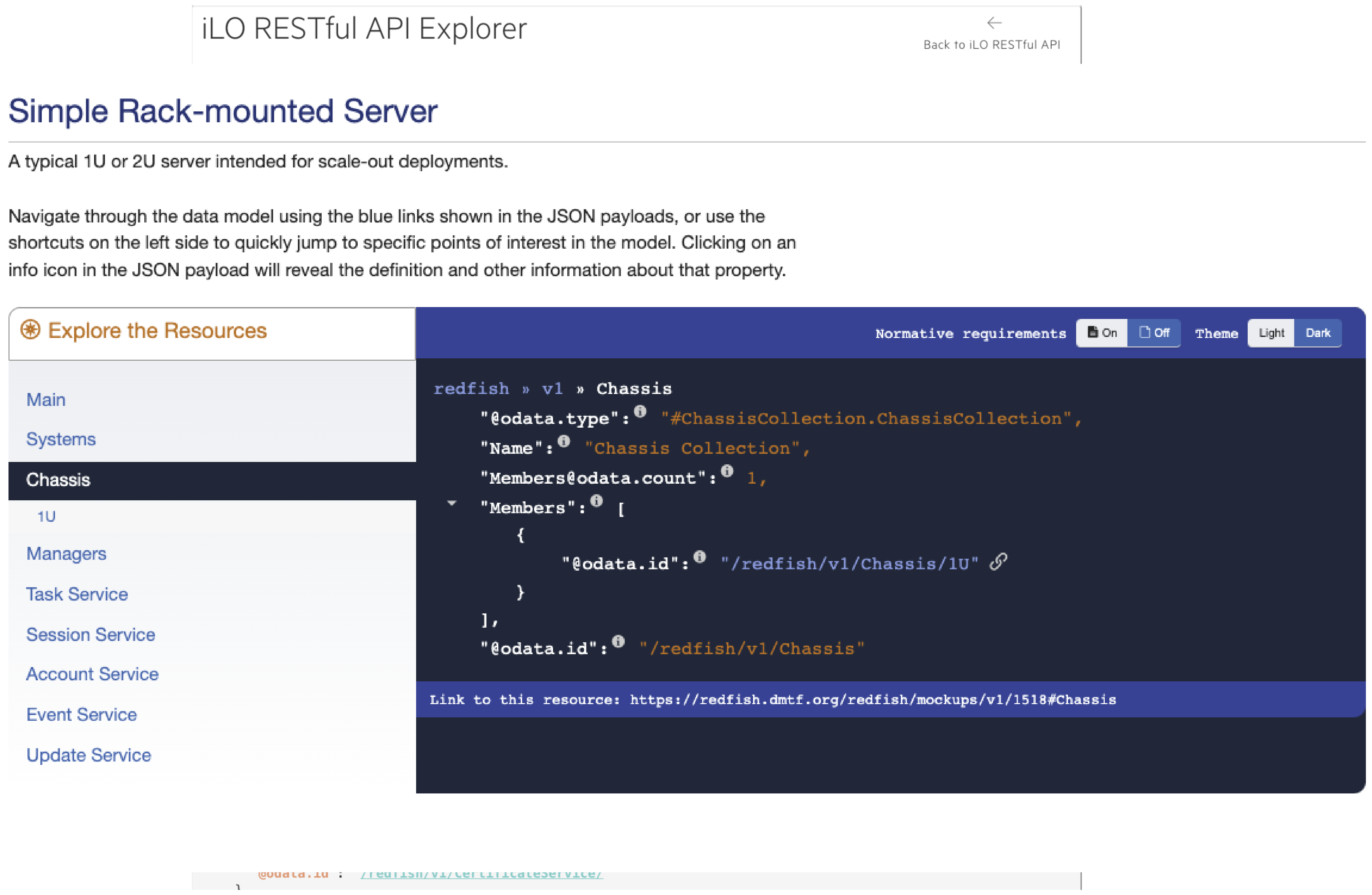
- To determine which session-based operations (GET, PATCH, POST, DELETE) are supported by a specific Redfish implementation, consult the official Redfish standard or manufacturer documentation.
- Different manufacturers may extend the standard Redfish data model, so implementations can vary.
- The DMTF has published mockups of hardware standards' data models, which can be viewed interactively. HPE also offers an interactive API viewer with ProLiant mockups.
  - https://ilorestfulapiexplorer.ext.hpe.com/
  - https://redfish.dmtf.org/redfish/v1

Setting Up a Development Environment:

- If Redfish-enabled hardware isn't available for testing, you can use mockup data models to simulate Redfish endpoints.
- DMTF provides a mockup emulator for static Redfish data models.
- HPE's Cray division offers an enhanced emulator with dynamic mockups that support advanced features like power control or session-based authentication.
- You can also create custom mockups using the DMTF's Redfish Mockup Creator tool.



```
{  "Headers":   {
     "X-XSS-Protection": "1; mode=block",
     "X-Content-Type-Options": "nosniff",
     "Transfer-Encoding": "chunked",
     "ETag": "W/"1799BBEB"",
     "Link": "</redfish/v1/SchemaStore/en/ServiceRoot.json/>; rel=describedby",
     "Allow": "GET, HEAD",
     "Cache-Control": "no-cache",
     "Date": "Thu, 11 Jul 2024 07:38:07 GMT",
     "OData-Version": "4.0",
     "X-Frame-Options": "sameorigin",
     "Content-type": "application/json; charset=utf-8"
   },
   "Response":   {
     "Systems":
     {
       "@odata.id": "/redfish/v1/Systems/"
     },
     "TelemetryService":
     {
       "@odata.id": "/redfish/v1/TelemetryService/"
     },
     "ComponentIntegrity":
     {
       "@odata.id": "/redfish/v1/ComponentIntegrity/"
     },
     "CertificateService":
     {
       "@odata.id": "/redfish/v1/CertificateService/"
```



iLO RESTful API Explorer

Back to iLO RESTful API

Simple Rack-mounted Server

A typical 1U or 2U server intended for scale-out deployments.

Navigate through the data model using the blue links shown in the JSON payloads, or use the shortcuts on the left side to quickly jump to specific points of interest in the model. Clicking on an info icon in the JSON payload will reveal the definition and other information about that property.

Explore the Resources

- Main
- Systems
- Chassis
  - 1U
- Managers
- Task Service
- Session Service
- Account Service
- Event Service
- Update Service

Normative requirements: On | Off — Theme: Light | Dark

redfish » v1 » Chassis

```
    "@odata.type": "#ChassisCollection.ChassisCollection",
    "Name": "Chassis Collection",
    "Members@odata.count": 1,
    "Members": [
        {
            "@odata.id": "/redfish/v1/Chassis/1U"
        }
    ],
    "@odata.id": "/redfish/v1/Chassis"
```

Link to this resource: https://redfish.dmtf.org/redfish/mockups/v1/1518#Chassis

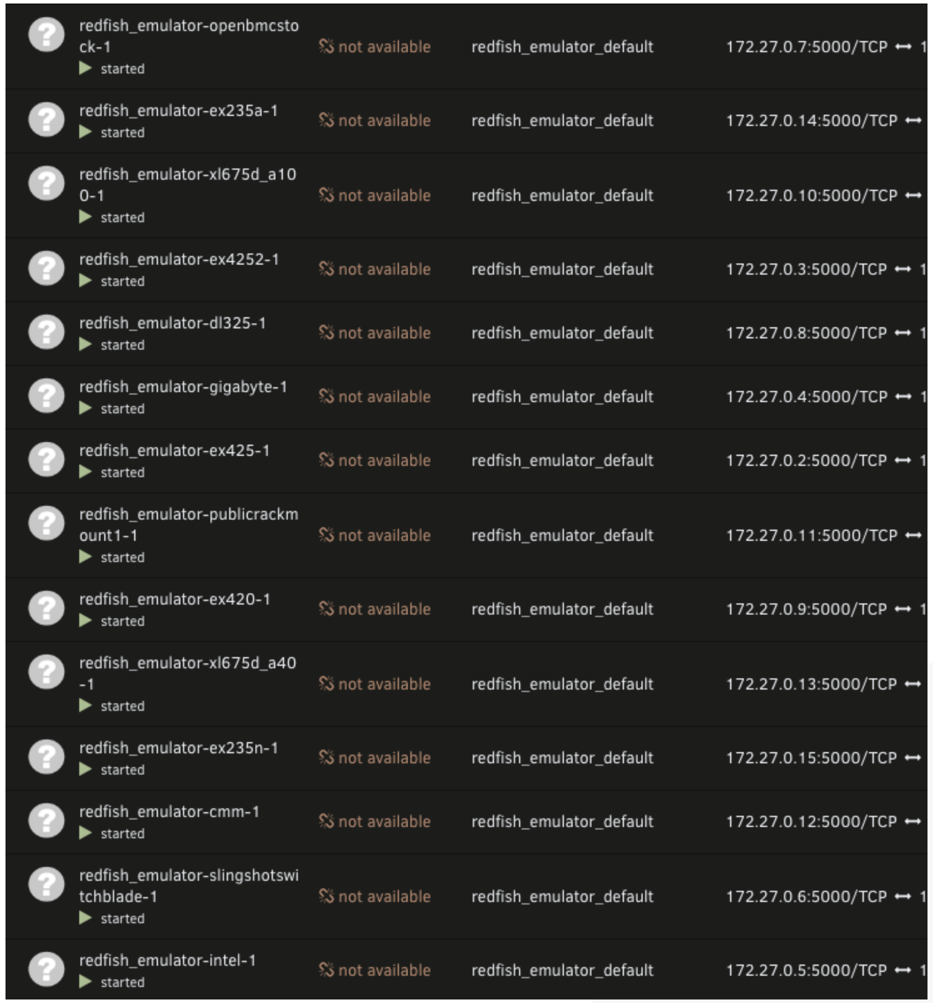
```
     "@odata.id": "/redfish/v1/CertificateService/"
  }
```

Building a Development Environment with HPE Cray's Open-Source Redfish Emulator

You can easily create a functional development environment without physical hardware using HPE Cray's open-source Redfish interface emulator. Available here!

Deployment Options:

- Virtual Environment: Run the emulator locally via Python in a virtual environment.
- Docker Container: Deploy as a Docker container for easier management.



redfish_emulator-openbmcstock-1 ▶ started	not available	redfish_emulator_default	172.27.0.7:5000/TCP ⟷ 1
redfish_emulator-ex235a-1 ▶ started	not available	redfish_emulator_default	172.27.0.14:5000/TCP ⟷
redfish_emulator-xl675d_a100-1 ▶ started	not available	redfish_emulator_default	172.27.0.10:5000/TCP ⟷
redfish_emulator-ex4252-1 ▶ started	not available	redfish_emulator_default	172.27.0.3:5000/TCP ⟷ 1
redfish_emulator-dl325-1 ▶ started	not available	redfish_emulator_default	172.27.0.8:5000/TCP ⟷ 1
redfish_emulator-gigabyte-1 ▶ started	not available	redfish_emulator_default	172.27.0.4:5000/TCP ⟷ 1
redfish_emulator-ex425-1 ▶ started	not available	redfish_emulator_default	172.27.0.2:5000/TCP ⟷ 1
redfish_emulator-publicrackmount1-1 ▶ started	not available	redfish_emulator_default	172.27.0.11:5000/TCP ⟷
redfish_emulator-ex420-1 ▶ started	not available	redfish_emulator_default	172.27.0.9:5000/TCP ⟷ 1
redfish_emulator-xl675d_a40-1 ▶ started	not available	redfish_emulator_default	172.27.0.13:5000/TCP ⟷
redfish_emulator-ex235n-1 ▶ started	not available	redfish_emulator_default	172.27.0.15:5000/TCP ⟷
redfish_emulator-cmm-1 ▶ started	not available	redfish_emulator_default	172.27.0.12:5000/TCP ⟷
redfish_emulator-slingshotswitchblade-1 ▶ started	not available	redfish_emulator_default	172.27.0.6:5000/TCP ⟷ 1
redfish_emulator-intel-1 ▶ started	not available	redfish_emulator_default	172.27.0.5:5000/TCP ⟷ 1

westMONROE

Building a Development Environment with HPE Cray's Open-Source Redfish Emulator

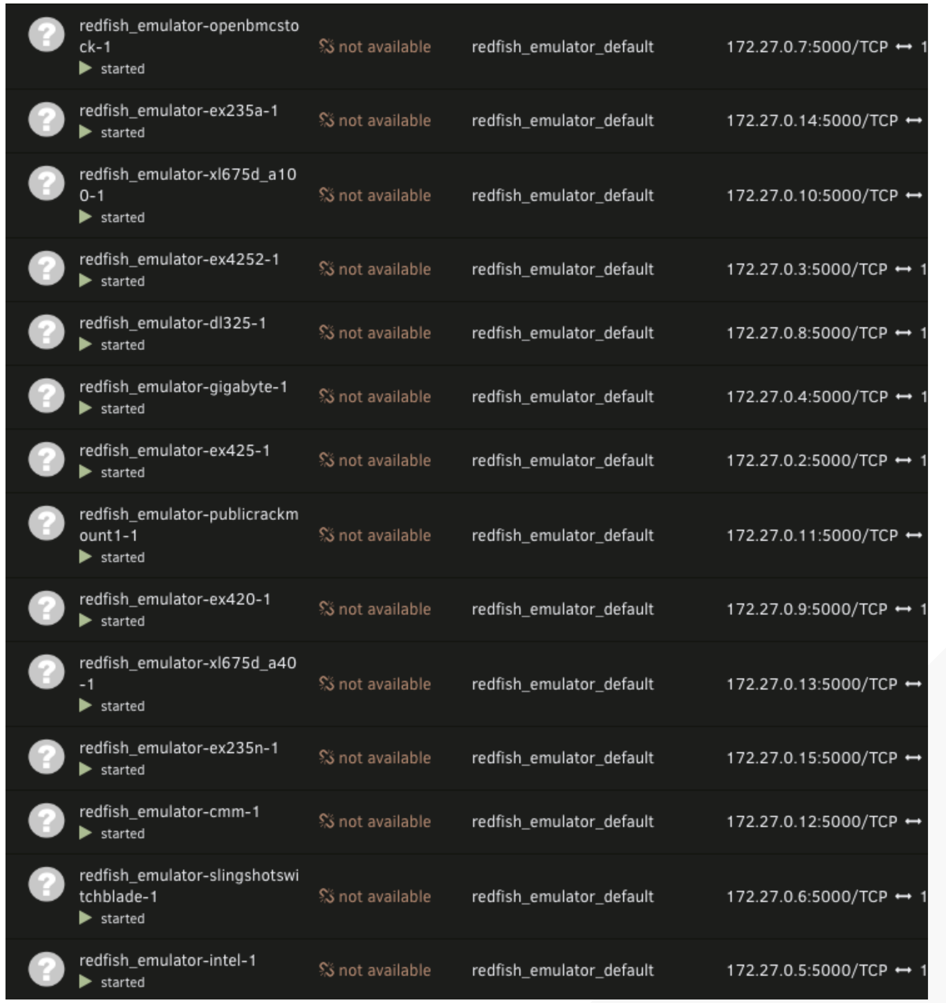
You can easily create a functional development environment without physical hardware using HPE Cray's open-source Redfish interface emulator. Available here!

Deployment Options:

- Virtual Environment: Run the emulator locally via Python in a virtual environment.
- Docker Container: Deploy as a Docker container for easier management.

Recommended for Multiple Mockups:

- Use Docker Compose to deploy numerous, diverse Redfish mockups simultaneously.



```
redfish_emulator-openbmcstock-1        not available   redfish_emulator_default   172.27.0.7:5000/TCP
  started
redfish_emulator-ex235a-1              not available   redfish_emulator_default   172.27.0.14:5000/TCP
  started
redfish_emulator-xl675d_a100-1         not available   redfish_emulator_default   172.27.0.10:5000/TCP
  started
redfish_emulator-ex4252-1              not available   redfish_emulator_default   172.27.0.3:5000/TCP
  started
redfish_emulator-dl325-1               not available   redfish_emulator_default   172.27.0.8:5000/TCP
  started
redfish_emulator-gigabyte-1            not available   redfish_emulator_default   172.27.0.4:5000/TCP
  started
redfish_emulator-ex425-1               not available   redfish_emulator_default   172.27.0.2:5000/TCP
  started
redfish_emulator-publicrackmount1-1    not available   redfish_emulator_default   172.27.0.11:5000/TCP
  started
redfish_emulator-ex420-1               not available   redfish_emulator_default   172.27.0.9:5000/TCP
  started
redfish_emulator-xl675d_a40-1          not available   redfish_emulator_default   172.27.0.13:5000/TCP
  started
redfish_emulator-ex235n-1              not available   redfish_emulator_default   172.27.0.15:5000/TCP
  started
redfish_emulator-cmm-1                 not available   redfish_emulator_default   172.27.0.12:5000/TCP
  started
redfish_emulator-slingshotswitchblade-1  not available   redfish_emulator_default   172.27.0.6:5000/TCP
  started
redfish_emulator-intel-1               not available   redfish_emulator_default   172.27.0.5:5000/TCP
  started
```

Building a Development Environment with HPE Cray's Open-Source Redfish Emulator

You can easily create a functional development environment without physical hardware using HPE Cray's open-source Redfish interface emulator. Available here!

Deployment Options:

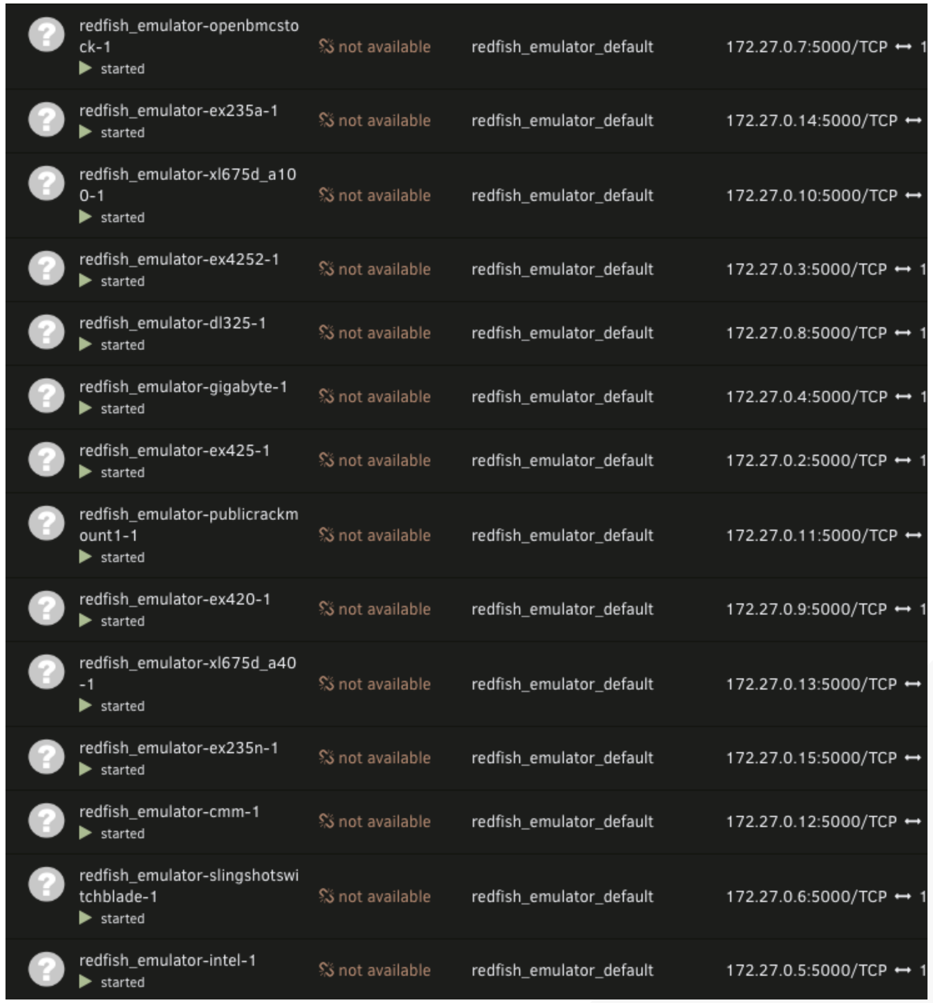
- Virtual Environment: Run the emulator locally via Python in a virtual environment.
- Docker Container: Deploy as a Docker container for easier management.

Recommended for Multiple Mockups:

- Use Docker Compose to deploy numerous, diverse Redfish mockups simultaneously.

Steps for Docker Compose Deployment:

- Modify the provided Dockerfile to use a public base image, such as alpine:3.14.
- Create a docker-compose.yml file, listing each container and its deployment parameters.
- *Fix issues with the dynamic authentication function.*
- Use docker-compose build to construct images, followed by docker-compose up to launch all containers.



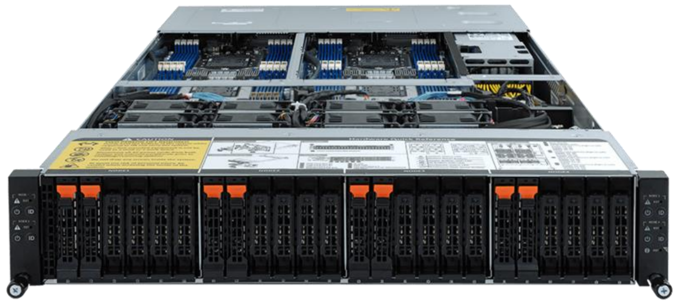
```
redfish_emulator-openbmcstock-1      not available   redfish_emulator_default   172.27.0.7:5000/TCP
▶ started
redfish_emulator-ex235a-1            not available   redfish_emulator_default   172.27.0.14:5000/TCP
▶ started
redfish_emulator-xl675d_a100-1       not available   redfish_emulator_default   172.27.0.10:5000/TCP
▶ started
redfish_emulator-ex4252-1            not available   redfish_emulator_default   172.27.0.3:5000/TCP
▶ started
redfish_emulator-dl325-1             not available   redfish_emulator_default   172.27.0.8:5000/TCP
▶ started
redfish_emulator-gigabyte-1          not available   redfish_emulator_default   172.27.0.4:5000/TCP
▶ started
redfish_emulator-ex425-1             not available   redfish_emulator_default   172.27.0.2:5000/TCP
▶ started
redfish_emulator-publicrackmount1-1  not available   redfish_emulator_default   172.27.0.11:5000/TCP
▶ started
redfish_emulator-ex420-1             not available   redfish_emulator_default   172.27.0.9:5000/TCP
▶ started
redfish_emulator-xl675d_a40-1        not available   redfish_emulator_default   172.27.0.13:5000/TCP
▶ started
redfish_emulator-ex235n-1            not available   redfish_emulator_default   172.27.0.15:5000/TCP
▶ started
redfish_emulator-cmm-1               not available   redfish_emulator_default   172.27.0.12:5000/TCP
▶ started
redfish_emulator-slingshotswitchblade-1  not available   redfish_emulator_default   172.27.0.6:5000/TCP
▶ started
redfish_emulator-intel-1             not available   redfish_emulator_default   172.27.0.5:5000/TCP
▶ started
```

PowerShell Script Demos

PowerShellRedfishTools on GitHub

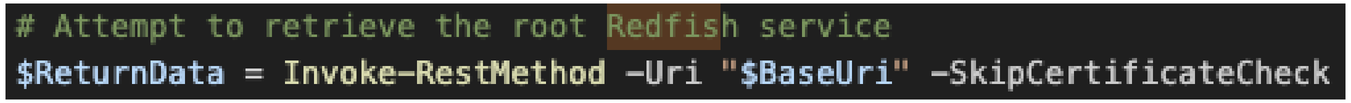
Overview of the Redfish Development Environment in Use Today

- **Redfish targets:**
  - *10.0.0.16 – "Ares"*
    - Physical HPE ProLiant DL360 Gen9
    - Redfish v1.0.0 on iLO 4 v2.79
  - *10.0.100.10:8466*
    - Emulated HPE ProLiant XL675d Gen10 Plus
    - Redfish v1.6.0 on iLO 5 v2.55
  - *10.0.100.10:8470*
    - Emulated GIGABYTE H262-Z63-00
    - Redfish v1.7.0 on AMI Redfish Server



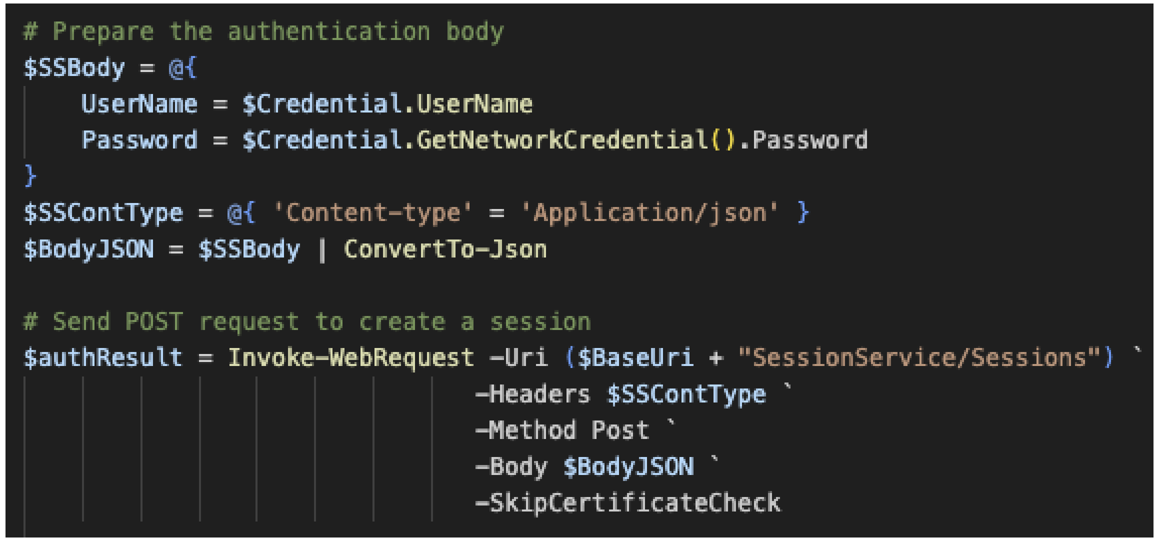
Overview of Key PowerShell Functions – Session Authentication

- The first function retrieves the Redfish root endpoint of the target and initializes key environment variables like BaseURI.



```
# Attempt to retrieve the root Redfish service
$ReturnData = Invoke-RestMethod -Uri "$BaseUri" -SkipCertificateCheck
```

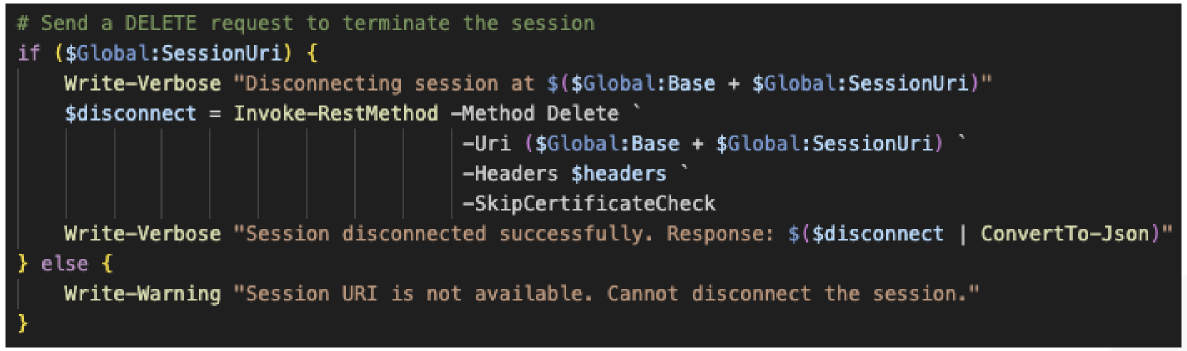
- The second function establishes a session by sending user credentials to the target via a POST request. Upon successful authentication, an XAuthToken is returned, which can be used for subsequent queries.



```
# Prepare the authentication body
$SSBody = @{
    UserName = $Credential.UserName
    Password = $Credential.GetNetworkCredential().Password
}
$SSContType = @{ 'Content-type' = 'Application/json' }
$BodyJSON = $SSBody | ConvertTo-Json

# Send POST request to create a session
$authResult = Invoke-WebRequest -Uri ($BaseUri + "SessionService/Sessions") `
                                -Headers $SSContType `
                                -Method Post `
                                -Body $BodyJSON `
                                -SkipCertificateCheck
```

- The final function closes the session by sending a DELETE request to the session's URI, effectively logging out of the target.



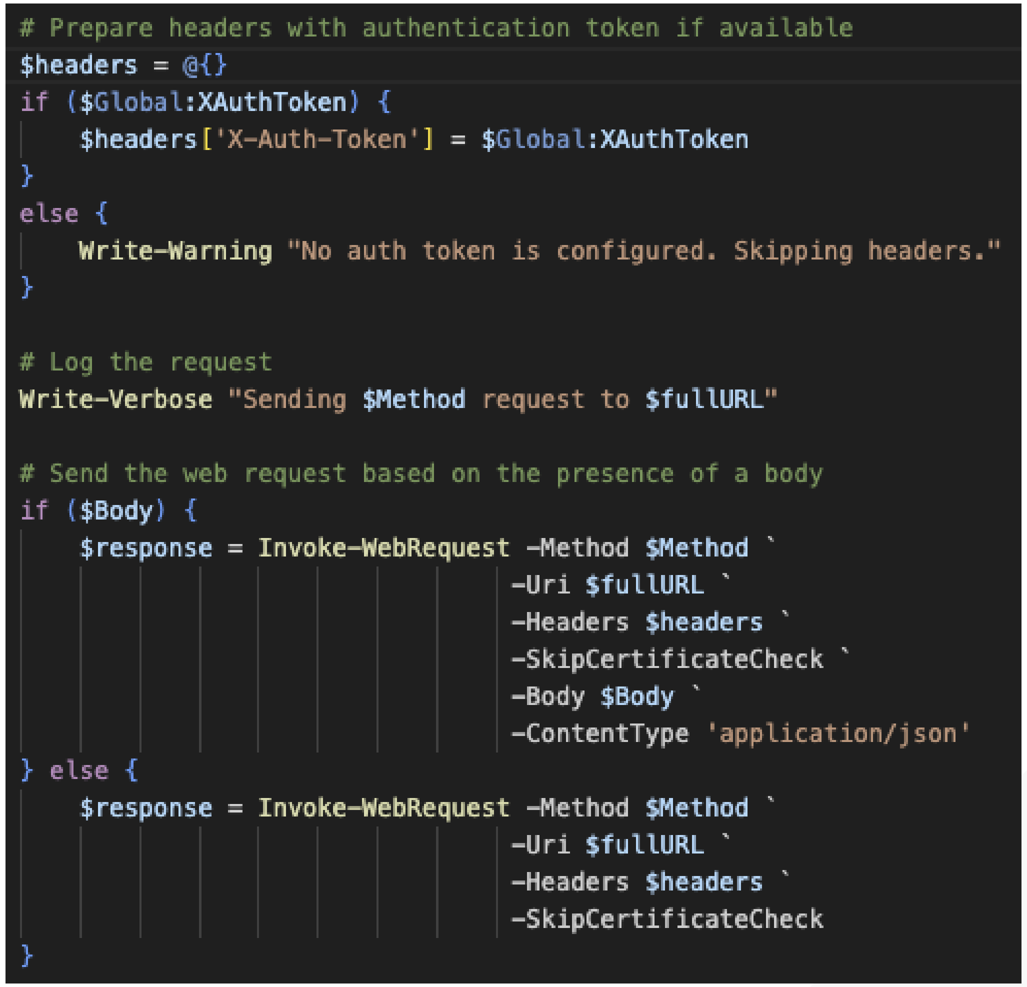
```
# Send a DELETE request to terminate the session
if ($Global:SessionUri) {
    Write-Verbose "Disconnecting session at $($Global:Base + $Global:SessionUri)"
    $disconnect = Invoke-RestMethod -Method Delete `
                                    -Uri ($Global:Base + $Global:SessionUri) `
                                    -Headers $headers `
                                    -SkipCertificateCheck
    Write-Verbose "Session disconnected successfully. Response: $($disconnect | ConvertTo-Json)"
} else {
    Write-Warning "Session URI is not available. Cannot disconnect the session."
}
```

westMONROE

Overview of Key PowerShell Functions – Invoking a Redfish Web Request

- To retrieve, modify, or remove configurations from the target system, the resource ID of the endpoint must first be identified. The XAuthToken is then used to authenticate and compile the web request.

- When performing PATCH or POST requests (for adding or updating configurations), a request body must be constructed in JSON format and passed into the function.


```
# Prepare headers with authentication token if available
$headers = @{}
if ($Global:XAuthToken) {
    $headers['X-Auth-Token'] = $Global:XAuthToken
}
else {
    Write-Warning "No auth token is configured. Skipping headers."
}

# Log the request
Write-Verbose "Sending $Method request to $fullURL"

# Send the web request based on the presence of a body
if ($Body) {
    $response = Invoke-WebRequest -Method $Method `
                                 -Uri $fullURL `
                                 -Headers $headers `
                                 -SkipCertificateCheck `
                                 -Body $Body `
                                 -ContentType 'application/json'
} else {
    $response = Invoke-WebRequest -Method $Method `
                                 -Uri $fullURL `
                                 -Headers $headers `
                                 -SkipCertificateCheck
}
```

Automating Data Center Inventory Collection

Use Case:

In many of our divestiture projects, we are tasked with migrating infrastructure from a parent company's domain to a new, independent environment. Often, the separating entity has limited or outdated documentation of their asset inventory. This tool facilitates the automated collection of infrastructure data across numerous target systems, enabling administrators to quickly gather detailed information. The custom reports generated can account for hardware variations and manufacturer-specific implementations, providing a comprehensive view of the environment.

Automating Data Center Inventory Collection

Use Case:

In many of our divestiture projects, we are tasked with migrating infrastructure from a parent company's domain to a new, independent environment. Often, the separating entity has limited or outdated documentation of their asset inventory. This tool facilitates the automated collection of infrastructure data across numerous target systems, enabling administrators to quickly gather detailed information. The custom reports generated can account for hardware variations and manufacturer-specific implementations, providing a comprehensive view of the environment.

Script Preparation:

- The properties that should be included in the inventory report must be defined in a JSON file, which is passed into the script file.
- **Properties**: Defines the key Redfish properties to extract from the API for each resource type.
- **FallbackProperties**: Specifies alternate properties to check if the primary ones are not available.
- **ResourceFilter**: Filters which Redfish resources to process based on URL patterns (e.g., /redfish/v1/Systems/*).
- **odataType**: Ensures the correct Redfish object type is processed, typically based on Redfish object schema

```
{
"SystemLog": {
        "Properties": {
                "BootOrder": ["Boot", "BootOrder"],
                "TotalSystemMemoryGiB": [
                            "MemorySummary",
                            "TotalSystemMemoryGiB"
                ]
        },
        "FallbackProperties": {
                "BootOrder":
                ["Boot","BootSourceOverrideSupported"]
        },
        "ResourceFilter": "/redfish/v1/Systems/*",
        "odataType": "#ComputerSystem.1.0.1.ComputerSystem"
        }
}
```

Automating Data Center Inventory Collection

Running the PowerShell Scripts:

1. Once the input JSON is prepared, execute Get-RecursiveRedfishLookup.ps1 to output a recursive view of the Redfish target:

```
$cred = Get-Credential

Get-RecursiveRedfishLookup.ps1 -TargetURIs 10.0.0.16,10.0.100.10:8466,10.0.100.10:8470 –Credential $cred
```

2. After the script has finished, execute Start-RedfishHostInventoryCollection.ps1 to build a report from the output.

```
Start-RedfishHostInventoryCollection.ps1 –RootDirectory “../Get-RecursiveRedfishLookup/10.0.0.16”
```

Script Usage:

```
Get-RecursiveRedfishLookup.ps1
    [-TargetURIs] <string[]>
    [-Credential] <PSCredential>
    [-redfishURLRoot <string>]
    [-OutputDirectory <string>]
    [-URLFilter <string>]
    [<CommonParameters>]
```

```
Start-RedfishHostInventoryCollection.ps1
    [-RootDirectory] <string>
    [-CSVOutputPath <string>]
    [<CommonParameters>]
```

Centralized System Log Collection for Faster Troubleshooting

Use Case:

When troubleshooting specific log events across multiple physical servers and datacenter equipment, manually accessing each server's BMC to extract logs can be time-consuming and inefficient. This tool simplifies the process by enabling centralized log collection across multiple targets. It supports various log formats and can be easily customized to handle different log types across diverse hardware and manufacturers.

Script Preparation:

- The properties that should be included in the log reports must be defined in a JSON file, which is passed into the script file.
- **Properties**: Defines the key Redfish properties to extract from the API for each resource type.
- **LogFormat**: Defines the structure that the properties are output in when printing logs to text files.

```
{
"#LogEntry.1.0.0.LogEntry": {
                "Properties": {
                                "Message": ["Message"],
                                "Severity": ["Severity"],
                                "Created": ["Created"],
                                "RecordId": ["RecordId"],
                                "Class": ["Oem", "Hp", "Class"]
                },
                "LogFormat": "[{Created}] ({Severity}) -       [{RecordId}] {Message}
[{Class}]"
}
```

Centralized System Log Collection for Faster Troubleshooting

Running the PowerShell Scripts:

1. Once the input JSON is prepared, execute Get-RedfishTargetLogs.ps1 to output all of the logs from the target system:

```
$cred = Get-Credential

Get-RedfishTargetLogs.ps1 -TargetURIs 10.0.0.16,10.0.100.10:8466,10.0.100.10:8470 –Credential $cred
```

Script Usage:

```
Get-RedfishTargetLogs.ps1
   [-TargetURIs] <string[]>
   [-Credential] <PSCredential>
   [-OutputDirectory <string>]
   [-PropertyMappingFile <string>]
   [-JSONOutput <bool>]
   [<CommonParameters>]
```

Executing Administrative Actions on Data Center Hardware

Use Case:

When invoking actions on different data center hardware using PowerShell, I frequently encountered differences in the data model that forced me to write custom functions for each device type. This was time-consuming and prone to errors. To address this, I developed a flexible solution that allows administrators to define their own operations in a JSON file, streamlining the process of managing hardware configurations.

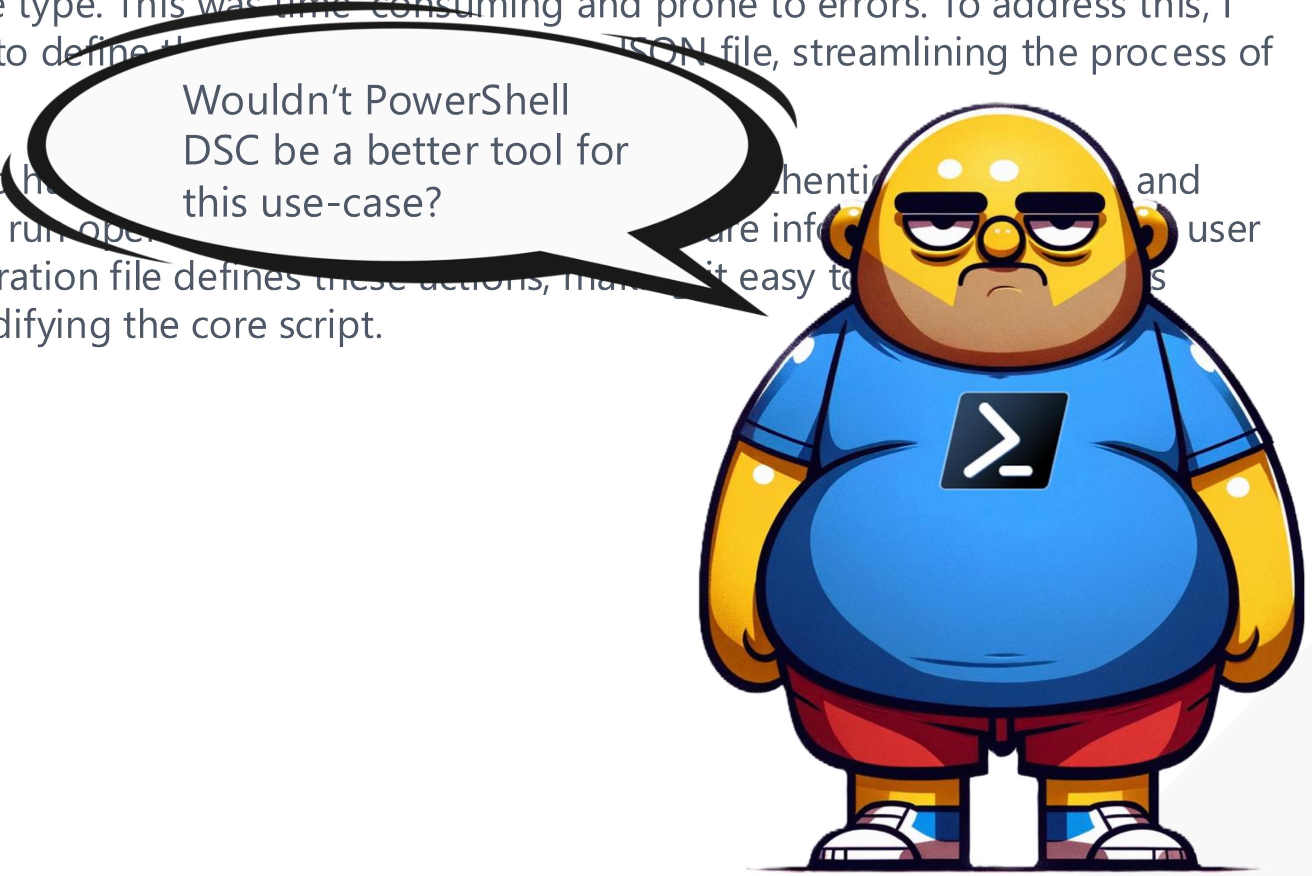
This script acts as a powerful framework that connects to hardware targets using the Redfish API, authenticates sessions, and executes user-defined actions. The script allows users to run operations such as retrieving hardware information, modifying user accounts, and deleting obsolete data. The JSON configuration file defines these actions, making it easy to extend the tool's functionality to different systems or devices without modifying the core script.

Executing Administrative Actions on Data Center Hardware

Use Case:

When invoking actions on different data center hardware using PowerShell, I frequently encountered differences in the data model that forced me to write custom functions for each device type. This was time-consuming and prone to errors. To address this, I developed a flexible solution that allows administrators to define th SON file, streamlining the process of managing hardware configurations.

This script acts as a powerful framework that connects to h hentic and executes user-defined actions. The script allows users to run oper are info user accounts, and deleting obsolete data. The JSON configuration file defines these actions, mak it easy to s functionality to different systems or devices without modifying the core script.



westMONROE

Executing Administrative Actions on Data Center Hardware

Script Preparation:

- The supported actions are defined in a JSON file, which is passed into the script file.
- **GetCommand** : This section defines the HTTP GET method, which retrieves the current state or values of the specified properties for a given resource.
- **SetCommand**: This block is used to modify or update a resource using a POST or PATCH method. The BodyTemplate is populated with user-provided values
- **DeleteCommand**: This operation sends a DELETE request to remove a resource.
- **Resource Filters**: These filters specify the Redfish endpoints to apply the operations.

```
"LocalUserAccountManagement": {
        "GetCommand": {
                "Method": "GET",
                "PropertyNames": [
                ["Id"],
                ["Oem", "Hpe", "Privileges"]]
        },
        "SetCommand": {
                "Method": "POST",
                "BodyTemplate": {
                        "UserName": "{{UserName}}",
                        "Password": "{{Password}}",
                        "RoleId": "{{RoleId}}"
                }
        },
        "DeleteCommand": {
                "Method": "DELETE"
        },
"GetResourceFilter": "/redfish/v1/AccountService/Accounts/*/",
"SetResourceFilter": "/redfish/v1/AccountService/Accounts/",
"DeleteResourceFilter": "/redfish/v1/AccountService/Accounts/*/"
}
```

Executing Administrative Actions on Data Center Hardware

Running the PowerShell Scripts:

1. Once the input JSON is prepared, execute Invoke-RemoteManagementActions.ps1 to execute actions against the target:

```
$cred = Get-Credential

Invoke-RemoteManagementActions.ps1 -TargetURI 10.0.0.16 –Credential $cred
```

Script Usage:

```
Invoke-RemoteManagementActions.ps1
   [-TargetURI] <string>
   [-Credential] <PSCredential>
   [<CommonParameters>]
```

Questions?