# Introduction and Overview of Redfish

**Patrick Boyd**
**Dell**

# Agenda

- Who is behind Redfish?
- What is Redfish?
- Simple Example
- Why HTTPS, REST, and JSON
- Slightly more complex example
- Discovery
- Creation example

# Who is behind Redfish?

## Promoter Companies


## Supporting Companies

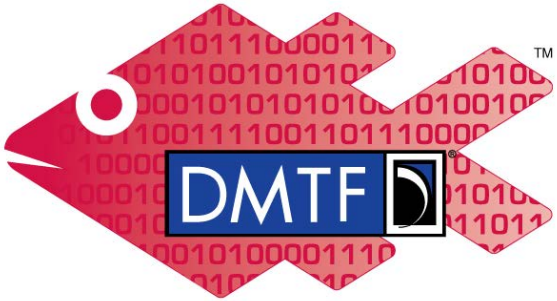
- AMI, Fujitsu, Huawei, IBM, Insyde Software, Mellanox, Microsemi, NetApp, Oracle, OSIsoft LLC, Quanta Computer, Seagate, Western Digital

## Industry Alliance Partners

- OpenCompute Project
- UEFI - Collaborating on Firmware Update and Host Interface work
- SNIA – Collaborating on Storage modeling / alignment between SSM and Redfish
- TGG – Pursuing relationship to work on Power/Cooling (existing DMTF Alliance Partner)

# What is Redfish?

- Industry Standard RESTful API for IT Infrastructure
  - HTTPS in JSON format based on Odata v4
  - Equally useable by Apps, GUIs, and Scripts
  - Schema-backed but human-readable



# What is Redfish?

- First release focused on Servers
  - A secure, multi-node capable replacement for IPMI-over-LAN
  - Add devices over time to cover customer use cases & technology
    - Direct attach storage, PCIe and SAS switching, NVDIMMs, Multifuction Adapters, Composability
  - Intended to meet OCP Remote Machine Management Requirements

# What is Redfish?

- Expand scope over time to rest of IT infrastructure
  - Working with SNIA to cover more advanced storage.
    - See “Overview of Swordfish” and “Swordfish Deep-dive” later in the conference
  - Plan on working with partners like Green Grid to cover Power/Cooling
  - Goal is to accommodate or map existing network switch standards over time

# Simple Example

Example Python code to retrieve serial number from a server:

```
rawData = urllib.urlopen('http://192.168.1.135/redfish/v1/Systems/1')
jsonData = json.loads(rawData)
print( jsonData['SerialNumber'] )
```

Output is:

```
1A87CA442K
```

***Example uses Redfish ComputerSystem resource, Authentication not shown**

# Simple Example

From the Linux command line

```
curl http://192.168.1.135/redfish/v1/Systems/1 | jq '.SerialNumber'
```

Output is:

```
"1A87CA442K"
```

***Example uses Redfish ComputerSystem resource, Authentication not shown**

****jq is a command line JSON parser available here https://stedolan.github.io/jq/**

# Why HTTPS, REST, and JSON?

- HTTPS: The Web Protocol
  - Well-understood by admins
  - Known security model
  - Known network configuration
- REST: The API Architecture
  - Has largely replaced SOAP for most web APIs
- JSON: The Data Format
  - Human-readable
  - Simpler than XML
  - Cross language support
- The combination of language support and ubiquity of REST, HTTP and JSON means that systems management tasks can be performed using the same skill set and tool chain as all other IT and dev/ops tasks.

# Slightly more complex example

- Using the same Computer System as the previous example, but this time we want to physically locate it in the data center. To do this we need to start the IndicatorLED on the computer system to “Blinking”.

```
echo {\"IndicatorLED\":\"Blinking\"} > led.json
curl -X PATCH --data “@led.json" -u login:password http://
192.168.1.135/redfish/v1/Systems/1
```

- Same paradigm for indicating a locally attached disk drive as well

```
echo {\"IndicatorLED\":\"Blinking\"} > led.json
curl -X PATCH --data “@led.json" -u login:password http://
192.168.1.135/redfish/v1/Systems/1/Storage/1/Drives/1
```

# Discovery

- The only well known URI for a Redfish Service is /redfish/
  - The contents should look like this:

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

  - This indicates that this Redfish Service supports v1 at the URI /redfish/v1

# Discovery

- Once the user has obtained the URI for the version they want, the user can obtain the Service Root.
  - The contents should something look like this:


```
{
        "@odata.context": "http://192.168.1.135/redfish/v1/$metadata#RootService",
        "@odata.id": "/redfish/v1/",
        "@odata.type": "#ServiceRoot.v1_1_0.ServiceRoot",
        "Id": "RootService",
        "Name":"Root Service",
        "RedfishVersion":"1.0.4",
        "UUID":"92384634-2938-2342-8820-489239905423",
        "Chassis": {"@odata.id":"/redfish/v1/Chassis"},
        "Mangers": {"@odata.id":"/redfish/v1/Managers"},
        “Systems": {"@odata.id":"/redfish/v1/Systems"}
}
```

# Discovery

- The service root contains the root level entries for a Redfish Service. Currently this may include:
  - Systems – A collection of all Application Servers
  - Chassis – A collection of all the Chassis
  - Managers – A collection of all the Managers
  - Task – A task management service
  - SessionService – A session management service
  - AccountService – An account management service
  - EventService – An event management and subscription service
  - Registries – A set of message registries
  - JsonSchemas – A set of JSON schema files describing the entities supported by this service
  - Fabrics – A collection of simple fabric models
  - UpdateService – A firmware update service

# Discovery

- Each of those entity types can then contain more entity types. For example ComputerSystem (under Systems from the Service Root) can contain:
  - Processors – A collection of all processors in the computer system
  - EthernetInterfaces – A collection of all Ethernet network interfaces
  - SimpleStorage – A collection of all very simple HBA style storage devices
  - LogServices – A collection of log services
  - Storage – A collection of more complex storage devices

# Discovery

- So let's take an example of discovering every Drive modeled by a Redfish Service. Drives are available under Storage which is under ComputerSystem.
  - So first we discover all ComputerSystems in our Redfish Service. From our ServiceRoot the URI was "/redfish/v1/Systems"

```
curl http://192.168.1.135/redfish/v1/Systems
```

```
{
  "@odata.context": "/redfish/v1/$metadata#ComputerSystemCollection",
  "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
  "Name": "Systems Collection",
  "Members@odata.count": 1,
  "Members": [
    {
      "@odata.id": "/redfish/v1/Systems/1"
    }
  ],
  "@odata.id": "/redfish/v1/Systems"
}
```

# Discovery

- In this example we have 1 ComputerSystem located at "/redfish/v1/Systems/1".
  - Now we need to know the URI for the StorageCollection of all Storage Devices hosted by this ComputerSystem

```
curl http://192.168.1.135/redfish/v1/Systems/1 | jq ".Storage"
```

```
{
  "@odata.id": "/redfish/v1/Systems/1/Storage"
}
```

  - In this example, and in fact quite often, the URI's are simply the name of the node added to the end of the parent URI, this is not guaranteed.

# Discovery

- Now we need to know how many storage entities are available on this computer system.

```
curl http://192.168.1.135/redfish/v1/Systems/1/Storage
```

```
{
  "@odata.context": "/redfish/v1/$metadata#StorageCollection",
  "@odata.type": "#StorageCollection.StorageCollection",
  "Name": “Storage Collection",
  "Members@odata.count": 2,
  "Members": [
    {
      "@odata.id": "/redfish/v1/Systems/1/Storage/SATA"
    },
    {
      "@odata.id": "/redfish/v1/Systems/1/Storage/RAID"
    }
  ],
  "@odata.id": "/redfish/v1/Systems/1/Storage"
}
```

# Discovery

- Now the user could query each Storage Entity determine the URI to each Drive entity and then query that, but there is a short cut.

```
curl
http://192.168.1.135/redfish/v1/Systems/1/Storage/SATA?$expand=Drives
| jq ".Drives"
```

```
[
    {
        "Id": "0",
        "Name": "Physical Disk 0",
        "IndicatorLED": "Lit",
        "Model": "12345",
        "Revision":"S20A",
        "CapacityBytes": 899527000000,
        …
    },
    {
        "Id": "1",
        "Name": "Physical Disk 1",
        "IndicatorLED": "Lit",
        "Model": "12345",
        "Revision":"S20A",
        "CapacityBytes": 899527000000,
        …
    },
    …
]
```

# Creation Example

- Obtaining data from the API and setting simple attributes is useful, but sometimes you need to do something more complex. Here is an example of how to create a Volume (Virtual Disk/Logical Disk) on an example system.

```
echo {\“Name\":\“Test Drive\“, \“CapacityBytes\”: 1099511627776,
\“VolumeType\”: \“Mirrored\”} > volume.json
curl -X POST --data “@volume.json" -u login:password http://
192.168.1.135/redfish/v1/Systems/1/Storage/RAID/Volumes
```

- This example just created a Volume named “Test Drive” with a size of 1TiB utilizing a mirror for redundancy.

# Backup

# Delete Volume

```
curl -X DELETE -u login:password http://
192.168.1.135/redfish/v1/Systems/1/Storage/RAID/Volumes/1
```

# Enable Encryption and Encrypt existing Volume

- First we need to give the Storage Node an Encryption Key to work with

```
echo {\“EncryptionKey\":\“Test Key\“} > key.json
curl -X POST --data “@key.json" -u login:password http://
192.168.1.135/redfish/v1/Systems/1/Storage/RAID/Actions/Storage.SetEn
cryptionKey
```

- Now Encrypt the Volume

```
echo {\“Encrypted\": true} > encrypt.json
curl -X PATCH --data “@encrypt.json" -u login:password http://
192.168.1.135/redfish/v1/Systems/1/Storage/RAID/Volumes/1
```

# Create Volume with Drives specified

- In our initial example we did not specify which drives to use in the Volume. This allows the Redfish Service to choose the drives. However, that may not be the drives desired by the client. Here is how to specify drives in the Volume creation.

```
echo {\“Name\":\“Test Drive\“, \“CapacityBytes\”: 1099511627776,
\“VolumeType\”: \“Mirrored\”, \“Links\”: {\“Drives\”:
[{\”@odata.id\”: \”/redfish/v1/Systems/1/Storage/RAID/Drives/1\”},
{\”@odata.id\”: \”/redfish/v1/Systems/1/Storage/RAID/Drives/2\”}]}} >
volume.json
curl -X POST --data “@volume.json" -u login:password http://
192.168.1.135/redfish/v1/Systems/1/Storage/RAID/Volumes
```