Staged Evolution of Integrating with Redfish
Interacting with hardware resources from a software perspective

Bryan Gartner
Sr. Technology Strategist
SUSE / bryan.gartner@suse.com
Redfish Integration / Usage

Agenda

- Introduction
- Redfish overview from an Open Source software person’s context
- Then, an evolving progression of
  - Accessing the Redfish API and Data Model contents
  - Start manipulating the target hardware to match what the overall use case requires
  - Leveraging all the pieces for an end-to-end deployment / solution
Overview of Redfish

From a software person’s context

- Yet another way to access a Baseboard Management Controller (BMC)
  - Bonus points
    - Superset of functionality compared to IPMI
    - Standardized approach across hardware partner platforms
- Provides / utilizes a REST API approach
  - Bonus points
    - Lots of possible ways to integrate
    - Composable, converged, hybrid-IT option to extend the software defined data center concept
    - Feels almost cloud-native like: a versioned API approach to manage the hardware that software lands upon
First steps
1st steps: accessing the API/Data Model

Start simple

- Via curl, interactive to scripted CLI walk through
  - literally started with a Google “linux redfish curl examples” search
    - Setup curl options
    - Validated access URL and credentials
    - Formatted output into readable (JSON)
    - Explored a subset of the data model
    - Scripted a poll across several systems
Accessing the Redfish API
man curl ;)

bwgartner@hpz210:/redfish> curl
> --silent \
> --insecure \
> --user admin \
> --header "Content-type: application/json" \
> --request GET \
> https://172.16.192.40/redfish/v1/
quiet mode
```
bwgartner@hpz210:~/redfish> curl \
> --silent \n> --insecure \n> --user admin \n> --header "Content-type: application/json" \n> --request GET \n> https://172.16.192.40/redfish/v1/ 
```
bwgartner@hpz210:~/redfish> curl \
> --silent \
> --insecure \
> --user admin \
> --header "Content-type: application/json" \
> --request GET \
> https://172.16.192.40/redfish/v1/
bwgartner@hpz210:/redfish> curl \
> --silent \
> --insecure \
> --user admin \
> --header "Content-type: application/json" \
> --request GET \
> https://172.16.192.40/redfish/v1/
bwgartner@hpz210:~/redfish> curl \
> --silent \n> --insecure \n> --user admin \n> --header "Content-type: application/json" \n> --request GET \n> https://172.16.192.40/redfish/v1/
bwgartner@hpz210:~/> curl \
> --silent \
> --insecure \
> --user admin \
> --header "Content-type: application/json" \
> --request GET \
> https://172.16.192.40/redfish/v1/
Use the Redfish top of API path and current version
Ok … worked … but output not entirely human readable
man jq ;)
Slice, filter, map and transform structured data
{
  "@odata.context": "/redfish/v1/$metadata#ServiceRoot",
  "@odata.id": "/redfish/v1",
  "@odata.type": "#ServiceRoot.1.0.0.ServiceRoot",
  "Oem": {},
  "Id": "",
  "Description": "",
  "Name": "Root Service",
  "RedfishVersion": "1.0.0",
  "UUID": "423c839f-f5e7-4081-1dbb-ac59ed46267f",
  "Links": {
    "Oem": {},
    "Sessions": {}
  },
  "Systems": {
    "@odata.id": "/redfish/v1/Systems"
  },
  "Chassis": {
    "@odata.id": "/redfish/v1/Chassis"
  },
  "Managers": {
    "@odata.id": "/redfish/v1/Managers"
  },
  "Tasks": {
    "@odata.id": "/redfish/v1/TaskService"
  },
  "SessionService": {
    "@odata.id": "/redfish/v1/SessionService"
  }
}
Cheat-sheet : 1/2 - Know you environment

BMC_IP=172.16.30.1
BMC_USER=ADMIN
BMC_PASS=ADMIN

# Install redfishtool (CLI)

git clone https://github.com/DMTF/Redfishtool.git

cd Redfishtool/ python3 redfishtool.py -r ${BMC_IP} -u ${BMC_USER} -p ${BMC_PASS} Systems -F

for BMC_IP in 10.0.1.11 10.0.1.12 10.0.1.13; do
    python3 redfishtool.py -r ${BMC_IP} -u ${BMC_USER} -p ${BMC_PASS} Systems -F | jq .SerialNumber
    python3 redfishtool.py -r $BMC_IP -u $BMC_USER -p $BMC_PASS Systems -F | jq .IndicatorLED

Done

python3 redfishtool.py -r $BMC_IP -u $BMC_USER -p $BMC_PASS Chassis list

python3 redfishtool.py -r $BMC_IP -u $BMC_USER -p $BMC_PASS Chassis -I 1

python3 redfishtool.py -r $BMC_IP -u $BMC_USER -p $BMC_PASS Chassis -I HA-RAID.0.StorageEnclosure.0

python3 redfishtool.py -r $BMC_IP -u $BMC_USER -p $BMC_PASS Systems -F | jq .UUID

python3 redfishtool.py -r $BMC_IP -u $BMC_USER -p $BMC_PASS Systems -F | jq .IndicatorLED

python3 redfishtool.py -r $BMC_IP -u $BMC_USER -p $BMC_PASS Chassis -I 1 setIndicatorLed Off

BMC_IP=$(dig +short node1.example.com)

unset https_proxy
Cheat-sheet : 2/2 - Game is opened

```bash
# get firmware versions
python3 redfishtool.py -r ${BMC_HOST} -u ${BMC_USER} -p ${BMC_PASS} Managers -F | jq .FirmwareVersion

# BMC
curl -s https://${BMC_IP}/redfish/v1/Managers/1 -k -u ${BMC_USER}:${BMC_PASS} | jq .FirmwareVersion

# BIOS python3 redfishtool.py -r ${BMC_HOST} -u ${BMC_USER} -p ${BMC_PASS} Systems -F | jq .FirmwareVersion

# System manufacturer
curl -s https://${BMC_IP}/redfish/v1/Systems/1 -k -u ${BMC_USER}:${BMC_PASS} | jq .Manufacturer

# System model
curl -s https://${BMC_IP}/redfish/v1/Systems/1 -k -u ${BMC_USER}:${BMC_PASS} | jq .Model

# get CPU information
curl -s https://${BMC_IP}/redfish/v1/Systems/1/Processors/1 -k -u ${BMC_USER}:${BMC_PASS} | jq .Model

# ram total
curl -s https://${BMC_IP}/redfish/v1/Systems/1 -k -u ${BMC_USER}:${BMC_PASS} | jq .MemorySummary.TotalSystemMemoryGiB

# ram modules
curl -k -u ${BMC_USER}:${BMC_PASS} -s https://${BMC_IP}/redfish/v1/Systems/1/Memory | jq ".Members | length"

# get BMC settings
curl -s https://${BMC_IP}/redfish/v1/Managers/1/EthernetInterfaces/2 -k -u ${BMC_USER}:${BMC_PASS} | jq .IPv4Addresses[0].Address

# get Health
curl -s https://${BMC_IP}/redfish/v1/Chassis/1 -k -u ${BMC_USER}:${BMC_PASS} | jq .Status.Health

# get IndicatorLED
curl -s https://${BMC_IP}/redfish/v1/Systems/1 -k -u ${BMC_USER}:${BMC_PASS} | jq .IndicatorLED

# fan mode
curl -s https://${BMC_IP}/redfish/v1/Managers/1/FanMode -k -u ${BMC_USER}:${BMC_PASS} | jq .Mode

# storage
curl -s https://${BMC_IP}/redfish/v1/Systems/1/SimpleStorage/1 -k -u ${BMC_USER}:${BMC_PASS} | jq .Devices[0].Model

# raid
curl -s https://${BMC_IP}/redfish/v1/Chassis/1/HA-RAID.0.StorageEnclosure.0 -k -u ${BMC_USER}:${BMC_PASS} | python -m json.tool

# power consumption
curl -s https://${BMC_IP}/redfish/v1/Chassis/1/Power/ -k -u ${BMC_USER}:${BMC_PASS} | jq .PowerControl[].PowerConsumedWatts

# BMC
curl -s https://${BMC_IP}/redfish/v1/Chassis/1/Power/ -k -u ${BMC_USER}:${BMC_PASS} | jq .PowerControl[].PowerMetrics.AverageConsumedWatts
```
Exploring the Data Model
Read authentication credentials from a file (tells curl to look for and use the .netrc file)
Grab sub-tree of data model
curl --silent --insecure --netrc --header "Content-type: application/json" --request GET https://172.16.192.40/redfish/v1/systems/1/ | jq | more
Simplified Scaling of Information Gathering
Wrap into a shell script

```bash
#!/bin/sh

IPSub="172.16"

for i in 192 195
do
  for j in 36 35 34 33 32
do
    echo "=== Node BMC - ${IPSub}.${i}.${j} ==="
    curl \ 
    --silent \ 
    --insecure \ 
    --netrc \ 
    --header "Content-type: application/json" \ 
    --request GET \ 
    https://${IPSub}.${i}.${j}/redfish/v1/Systems/1/ \ 
    | jq '{Model}'
  done
done
```

~ ~ ~ ~
Loop through several BMC IP ranges

```bash
#!/bin/sh

IPSub="172.16"

for i in 192 195
do
  for j in 36 35 34 33 32
done

echo "=== Node BMC - ${IPSub}.${i}.${j} ==="
curl \
  --silent \
  --insecure \
  --netrc \
  --header "Content-type: application/json" \
  --request GET \
  https://${IPSub}.${i}.${j}/redfish/v1/Systems/1/ \
  | jq '{Model}'
done
```

#! /bin/sh

IPSub="172.16"

for i in 192 195
do
  for j in 36 35 34 33 32
do
    echo "=== Node BMC - ${IPSub}.${i}.${j} ==="
curl \
    --silent \ 
    --insecure \ 
    --netrc \ 
    --header "Content-type: application/json" \ 
    --request GET \ 
    https://${IPSub}.${i}.${j}/redfish/v1/Systems/1/ \ 
    | jq '.{Model}"
  done
done

~ ~ ~ ~
Other possible calls

Of course, a lot more ways this can be also exercised

- Redfish API
- Exploring Data Model
  - Redfish Developer Hub (see Mockups)
- Programmatic Interfaces
  - Language bindings: C, Javascript, Powershell, Python, Ruby, ...
  - DevOps: Ansible, Chef, Nagios, Puppet, ...
Additional references

Homework exercises left for the reader

- Dell-related
  - Knowledge Base - Redfish
- Fujitsu
  - iRMC Redfish API Specifications
  - Redfish White Paper
- HPE-related
  - iLO RESTful API
  - iLO RESTful API Explorer
- Intel
  - Redfish, RESTful and x-UEFI
- Lenovo-related
  - xClarity Controller Redfish REST API
- Supermicro
  - Server Management (Redfish API)
- ...
2nd step
Understand the target

Helping the hardware-challenged (aka software folks)

· Beyond the on-line Mockups ...
  – Visit GitHub openStack/python-redfish
    · git clone
      · Install a container run-time engine
      · In dmtf/mockup*, build, run, use the container
  – Homework left as an exercise for the reader
    · You can install (from src, PyPi, or packages the redfish-client)
New tools

Other techniques and/or target resources ...

**SUSE Manager / Uyuni**
Opensource software management solution
Leverages Saltstack, and starting development of a Redfish integration - openSUSE/redfish
Query/select/configure + de-configure/de-select/return to a known state
The hardware needed to match the desired software workloads as part of the overall deployment lifecycle
salt-call redfish.set_property IndicatorLED “Blinking” … (or “Off”)

**Terraform**
Starting to leverage this technology, which matches quite well with the underlying infrastructure
restapi provider to interact with Redfish
terraform-provider-oneview overlay that works with the HPE Composable Infrastructure APIs
More choices

Continually exploring some new and some existing options

- In the end, the true value proposition of open source for users is “freedom of choice”
  - So with the trends of
    - Software-Defined Infrastructure
    - Migration to Infrastructure-as-Code
    - Cloud-Native computing principles (everything is really an API/version)
  - Providing choices in each matrix element and layer approach is highly desirable
The Bento Project
Bento : manage end-to-end deployment
Hardware: HPE Apollo 2000 + 4 x XL170r

Rack your servers then connect power & network
First / BMC: update & setup the iLO interfaces
Redfish: BIOS’ easy mass setup

Second / BIOS: Date and time, performance mode, CPU & Memory tweaking, disks allocation, boot sequence…
Redfish: Ceph’s easy mass controllers setup

Third / Disks Controllers: Without Redfish: (1 x RAID-0 per drive) x 24 = PAIN
Redfish: 1 x RAID-0 per drive in a « for » loop = EASY

Fourth / Gathering data: Mainly MAC Addresses and servers’ resources
Redfish: Thank you

Redfish usage for this deployment is done.
It will be back for platform monitoring and lifecycle.

We can now use our scripts and software automation for:
Bare-metal automated deployment with a prepared USB key > Each node becomes a SLES KVM
KVM automation > Nodes are populated with VMs enveloppes using a CSV file
NTP / DNS / DHCP setup > Each node gets a VM deployed for such a role
Ceph cluster deployment > Using VMs (careful, support warning!)
Kubernetes cluster & registry deployment > Linked to the Ceph cluster
(optional) Cloud Foundry deployment > Based on kubernetes deployment
Summary

So interesting to explore / discover / leverage

- Redfish integration is an ever expanding utility / frontier
- Allows boundary crossing from developers to operations and across the classic IT silos
  — Game Meet On!
Thank You
Unpublished Work of SUSE LLC. All Rights Reserved.
This work is an unpublished work and contains confidential, proprietary and trade secret information of SUSE LLC. Access to this work is restricted to SUSE employees who have a need to know to perform tasks within the scope of their assignments. No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of SUSE. Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.

General Disclaimer
This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. SUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for SUSE products remains at the sole discretion of SUSE. Further, SUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All SUSE marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.