
DELL EMC
Make It Real
DIGITAL TRANSFORMATION

Using Ansible and Redfish to automate systems management

Jose Delarosa
May 9, 2018

DELL EMC

Before we start

- Thank you for coming to this session
- Please ask questions: It’s OK to interrupt
- If time runs out, happy to talk to you afterwards

Who am I?

- Linux Engineer
- Part time technology evangelist
- Part-time systems engineer
- Part-time developer
- @jdelaros1

Why are we all here?

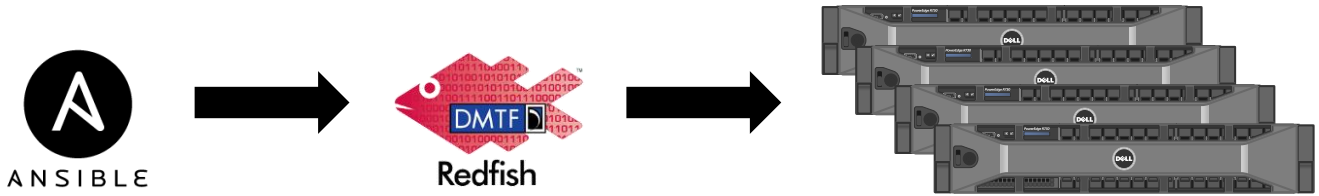
- Wrote some code using some really cool tools that will make your life easier.
- If you manage servers (i.e. sysadmin, SRE) in a lab or data center, this is for you.
- If you are an open source developer in IT, this is for you.
- If you like experimenting with new tools, this is definitely for you!

Motivation

- I had a need to scale OOB management
- I had a need to automate OOB management
- Needed to be open source
- Could have used shell scripting for some of it, but wanted something different

Agenda

1. Out-of-Band Management
2. Redfish (scalability)
3. Ansible (automation)
4. Scalability + Automation =



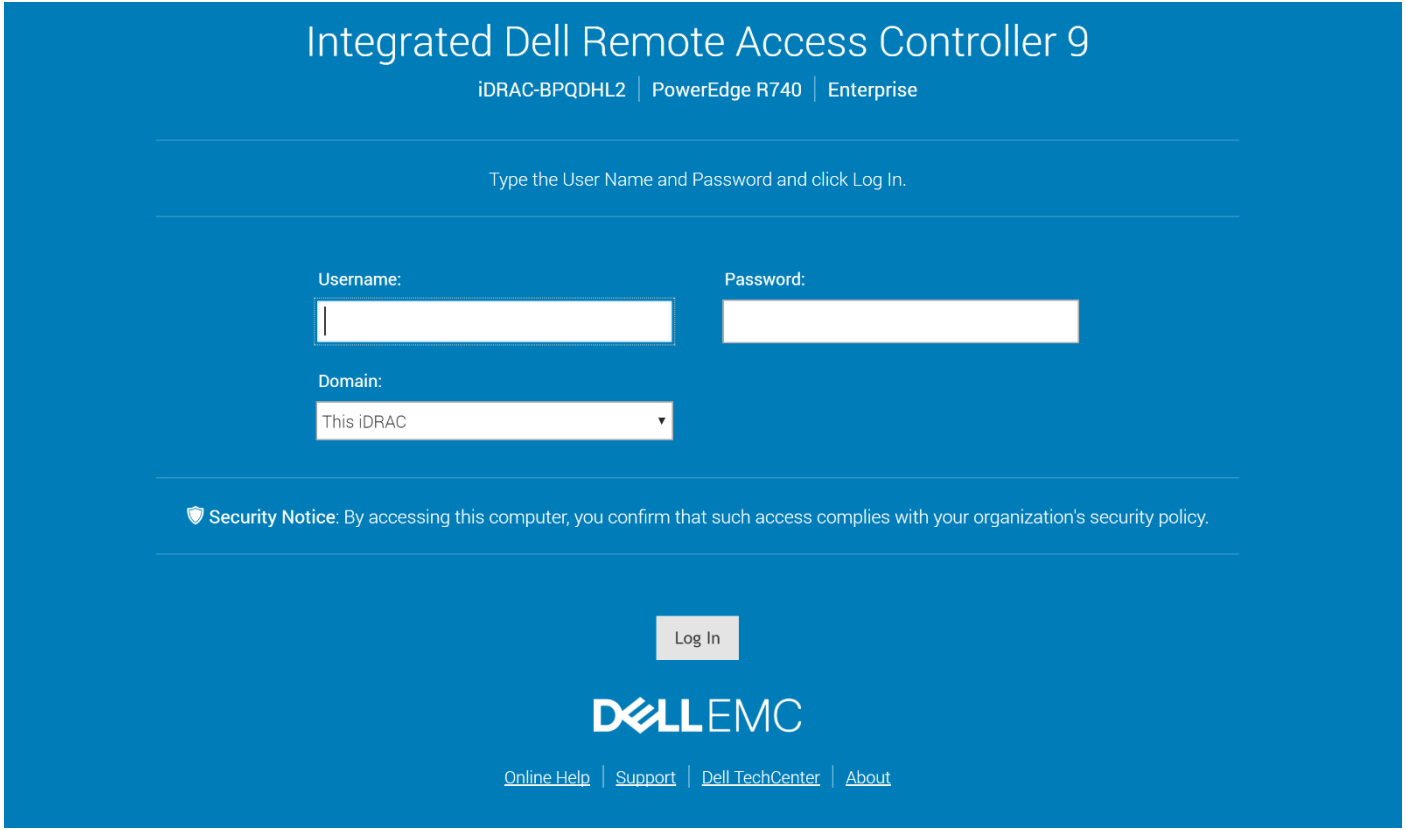
Out-of-Band Management Overview

DELL EMC

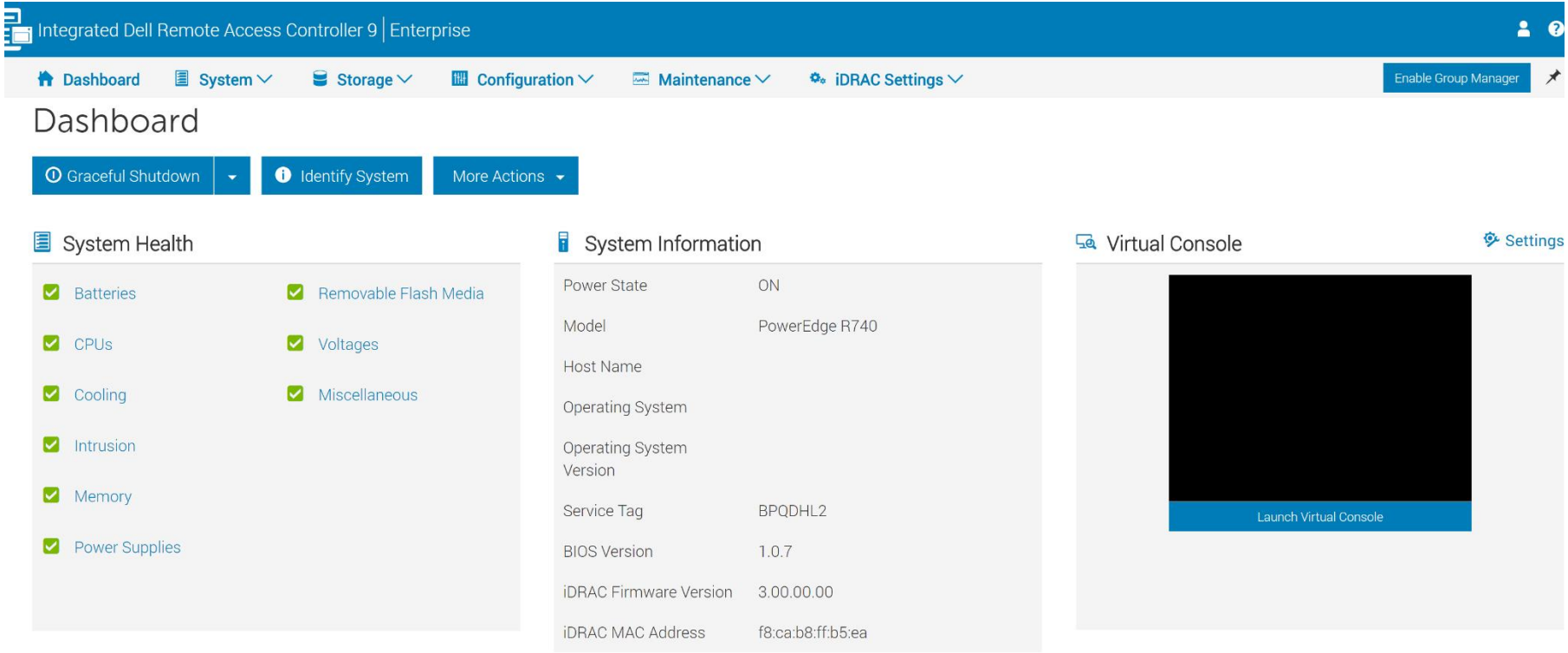
What is Out-of-Band (OOB) management?

- Server management independent of the server's operating system and main power
- Provided by an embedded chip, has its own Ethernet port, usually connected to a separate management network
- Goes by many names: iDRAC, iLO, IMM, BMC
- Management includes:
  - Device inventory
  - Hardware failure detection
  - System event logs
  - BIOS configuration

Login

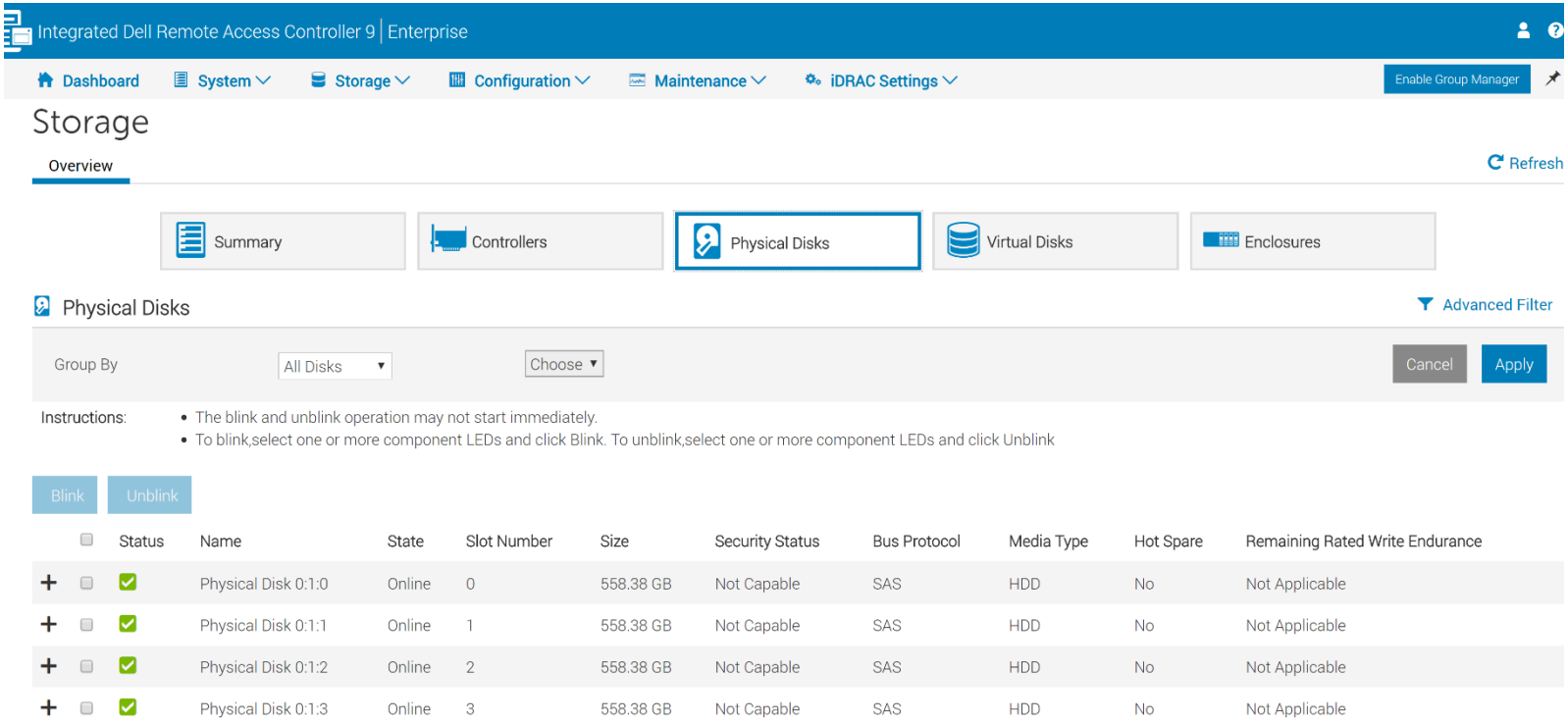

Integrated Dell Remote Access Controller 9
iDRAC-BPQDHL2 | PowerEdge R740 | Enterprise
Type the User Name and Password and click Log In.
Username:
Password:
Domain:
This iDRAC
Security Notice: By accessing this computer, you confirm that such access complies with your organization's security policy.
Log In
DELL EMC
Online Help | Support | Dell TechCenter | About

Dashboard

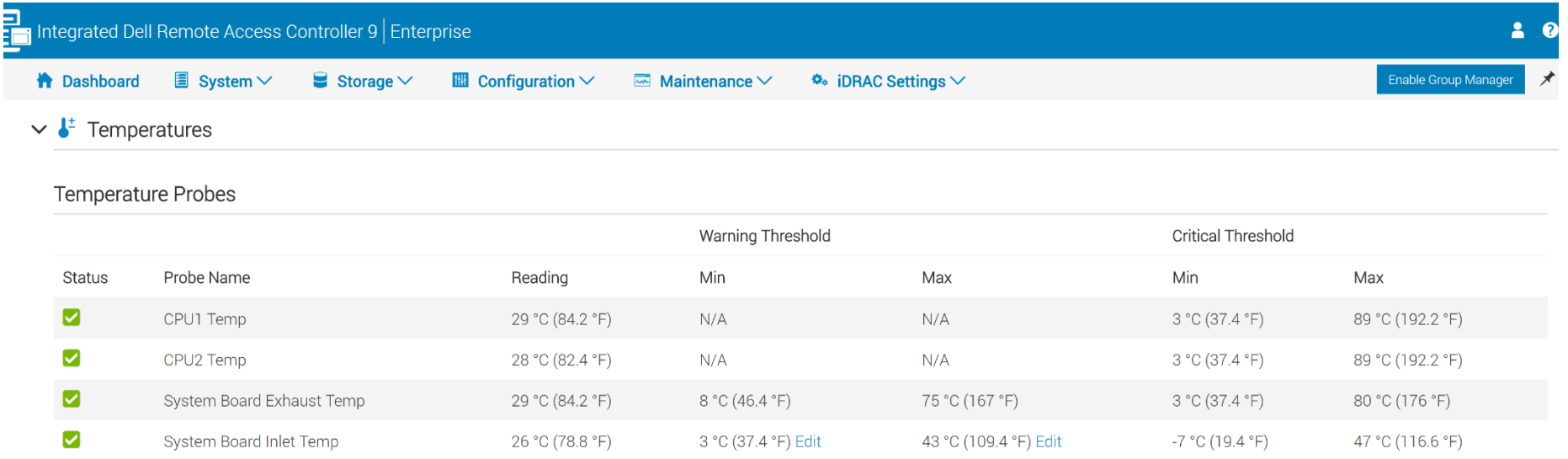


DELL EMC

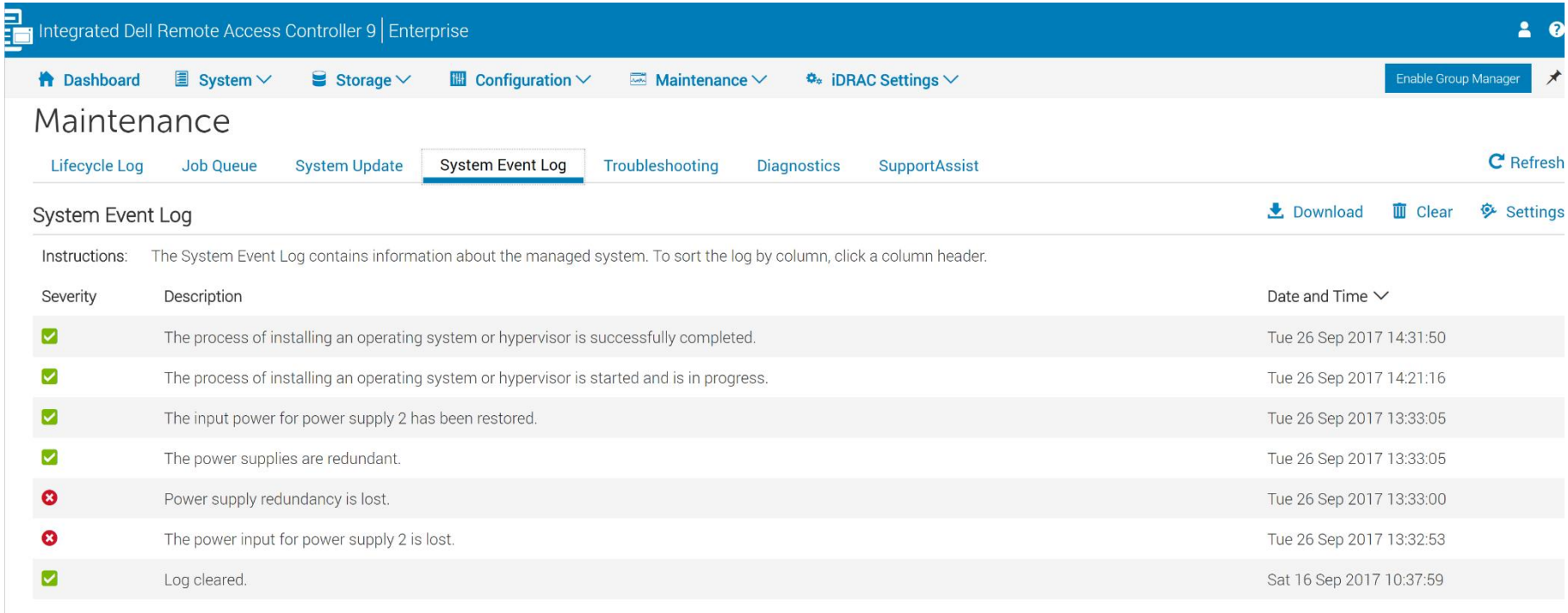
Hard Drives



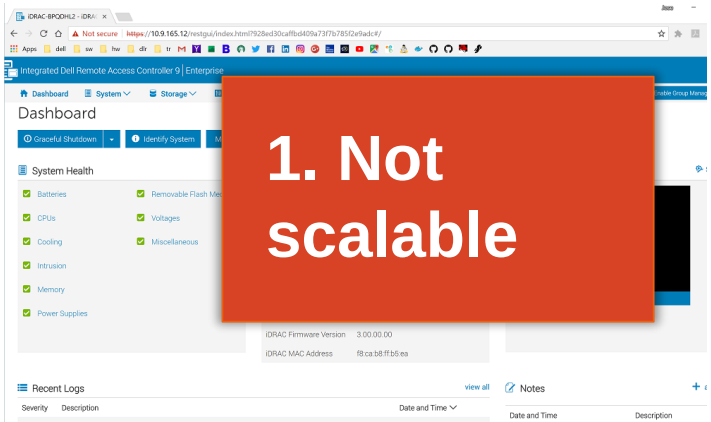
Thermal



System Event Logs



Simple OOB Management



1. Not scalable

2. Not automated

Redfish Overview

DELL EMC

Redfish Overview

- Open source, open industry standard specification published by the DMTF for hardware management.
- Provides a RESTful API used to obtain information and exert control over servers via an OOB controller.
- Built on a modern tool-chain which includes HTTPS, JSON and the OData standard.
- A Redfish request is sent as an URI, so a client could be any application on a server, workstation or mobile device.

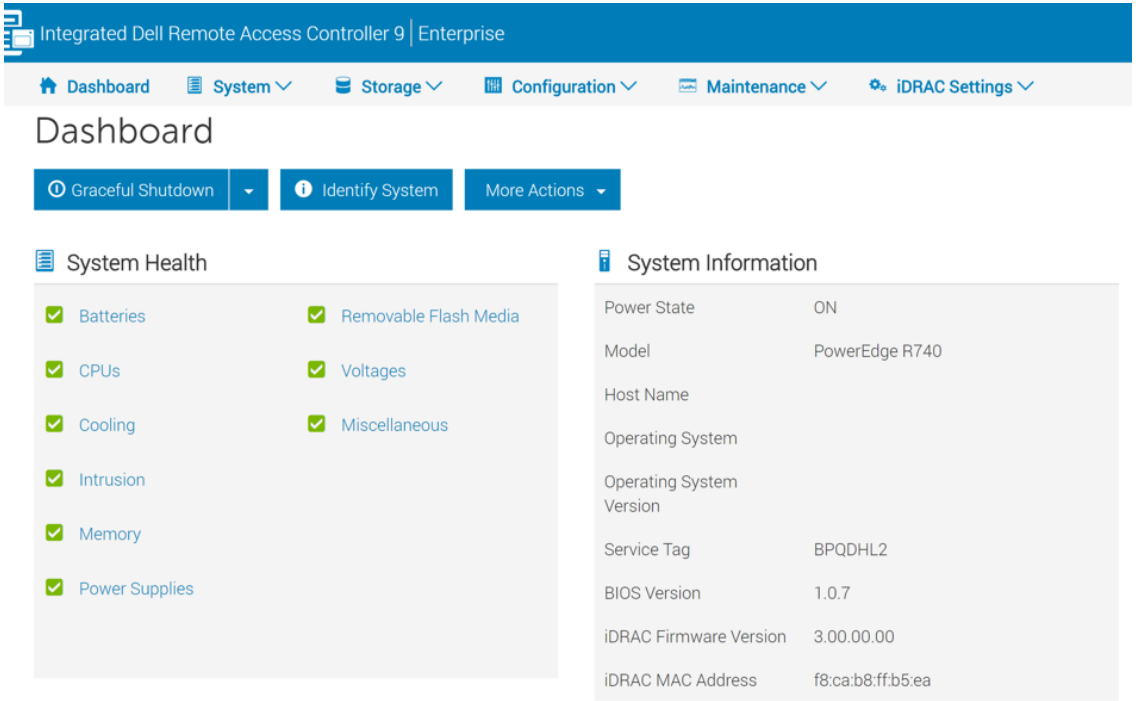
What can we do with Redfish?

- Retrieve server health status
- Retrieve hardware and firmware inventory
- Power up, power down, warm boot, cold boot
- Chan
- Chan
- Configure OOB controller (i.e. users, network settings)
- Configure RAID
- Firmware updates

https://www.dmtf.org/standards/redfish

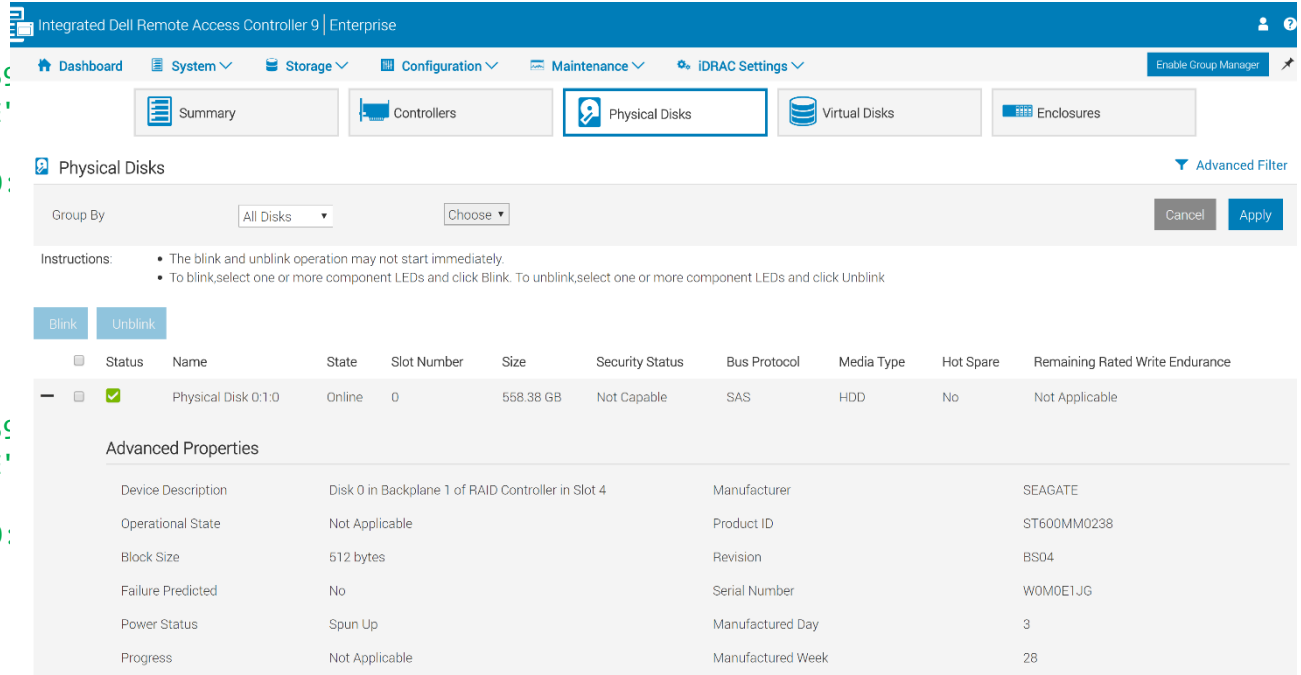
Example: System Health

```
$ curl https://<OOB>/redfish/v1/Systems/System.Embedded.1 --user root:password | jq .Status
{
  "Health": "OK",
  "HealthRollUp": "OK"
}
```



Example: Hard Drives

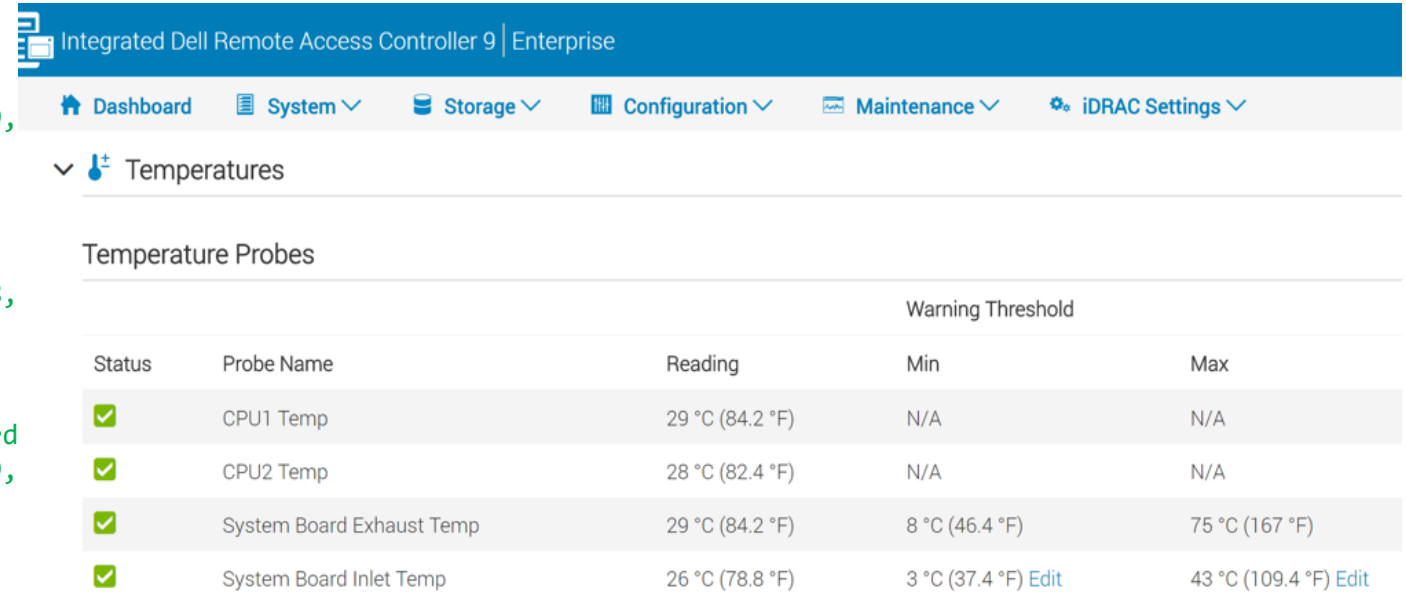
```
$ curl https://<OOB>/redfish/v1/Systems/System.Embedded.1/Storage/Controllers/RAID.Slot.4-1
 --user root:password | jq .Devices
[
  {
    "CapacityBytes": 5995505
    "Manufacturer": "SEAGATE"
    "Model": "ST600MM0238",
    "Name": "Physical Disk 0
    "Status": {
      "Health": "OK",
      "HealthRollup": "OK",
    }
  },
  {
    "CapacityBytes": 5995505
    "Manufacturer": "SEAGATE"
    "Model": "ST600MM0238",
    "Name": "Physical Disk 0
    "Status": {
      "Health": "OK",
      "HealthRollup": "OK",
    }
  },
```



Example: Thermal

```
$ curl https://<OOB>/redfish/v1/Chassis/System.Embedded.1/Thermal --user root:password | jq
'.Temperatures[] | {name:.Name, readingCelsius: .ReadingCelsius, health: .Status.Health}'
```

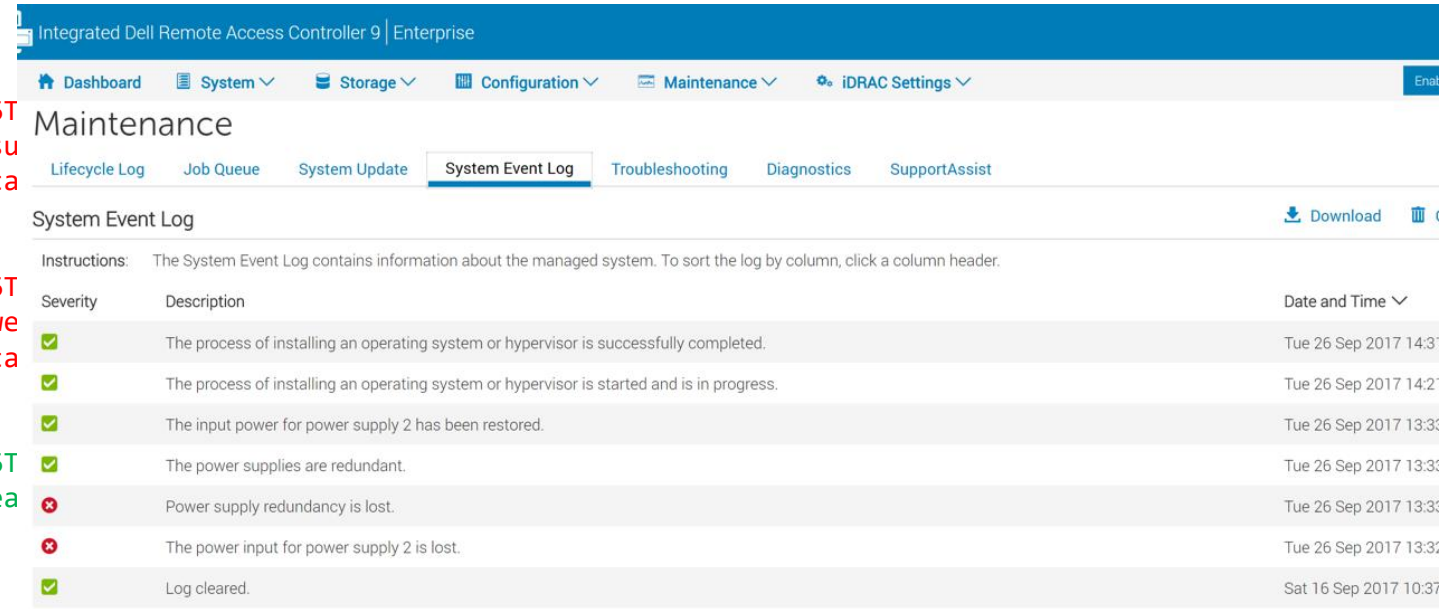
```
{
  "name": "CPU1 Temp",
  “readingCelsius": 29,
  "health": "OK"
}
{
  "name": "CPU2 Temp",
  “readingCelsius": 28,
  "health": "OK"
}
{
  "name": "System Board
  “readingCelsius": 29,
  "health": "OK"
}
..
```



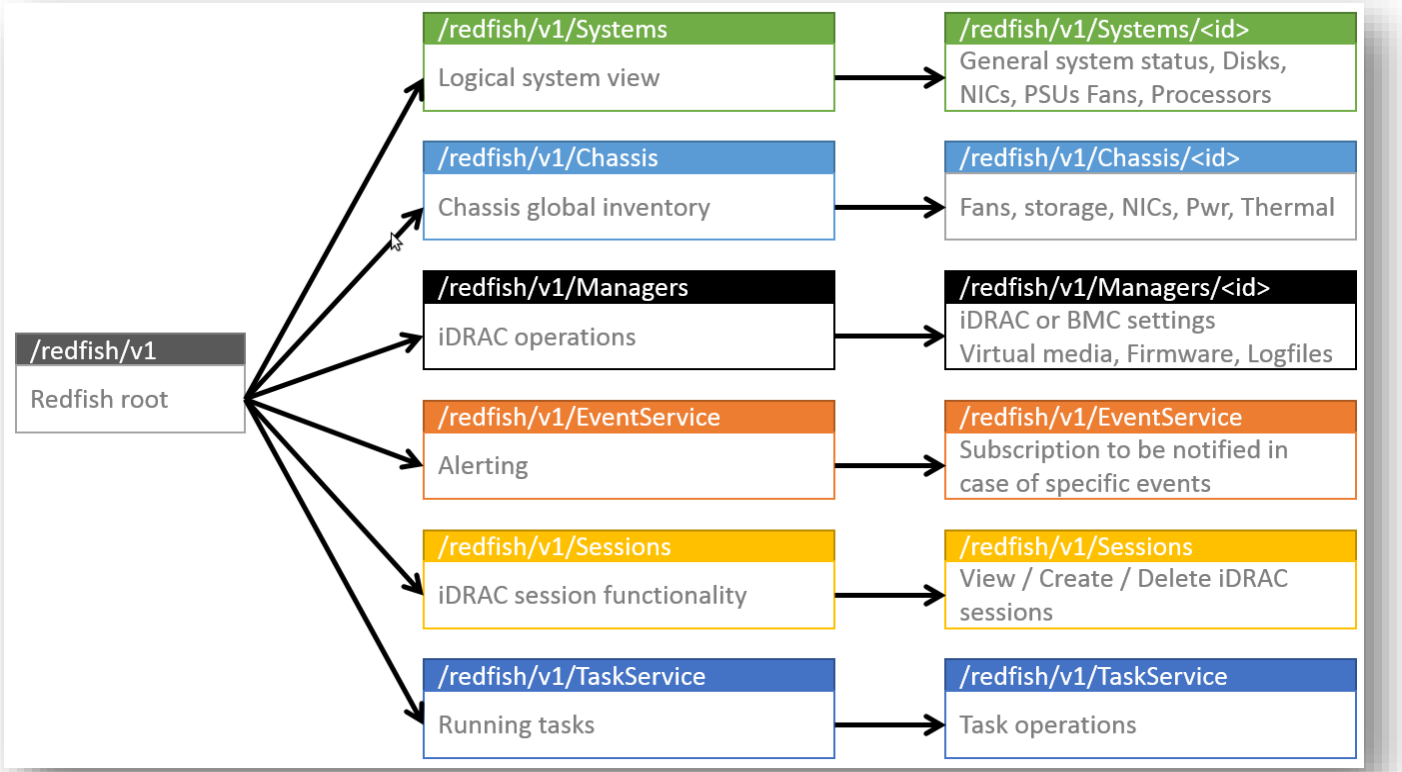
Example: System Event Logs

```
$ curl https://<OOB>/redfish/v1/Managers/iDRAC.Embedded.1/Logs/Sel --user root:password | jq
'.Members[] | {date: .Created, message: .Message, severity: .Severity}'
```

```
--- snip ---

{
  "date": "2017-09-26T
  "message": "Power su
  "severity": "Critica
}
{
  "date": "2017-09-26T
  "message": "The powe
  "severity": "Critica
}
{
  "date": "2017-09-16T
  "message": "Log clea
  "severity": "Ok"
}
```



Redfish API tree structure



System APIs

Redfish API URIs
/redfish/v1
/redfish/v1/Systems
/redfish/v1/Systems/<ServiceTag+nodeid>
/redfish/v1/Systems/System.Embedded.1/Actions/ComputerSystem.Reset
/redfish/v1/Systems/System.Embedded.1/Bios
/redfish/v1/Systems/System.Embedded.1/BootSources
/redfish/v1/Systems/System.Embedded.1/Processors
/redfish/v1/Systems/System.Embedded.1/Processors/<Processor-FQDD>
/redfish/v1/Systems/System.Embedded.1/EthernetInterfaces
/redfish/v1/Systems/System.Embedded.1/EthernetInterfaces/<EthernetInterface-FQDD>
/redfish/v1/Systems/System.Embedded.1/EthernetInterfaces/<EthernetInterface-FQDD>/Vlans
/redfish/v1/Systems/System.Embedded.1/Storage/Controllers
/redfish/v1/Systems/System.Embedded.1/Power/PowerSupplies
/redfish/v1/Systems/System.Embedded.1/SecureBoot
/redfish/v1/Systems/System.Embedded.1/Sensors/Fans

Chassis APIs

Redfish API URIs
/redfish/v1/Chassis
/redfish/v1/Chassis/System.Embedded.1
/redfish/v1/Chassis/System.Embedded.1/Thermal
/redfish/v1/Chassis/System.Embedded.1/Sensors/Fans
/redfish/v1/Chassis/System.Embedded.1/Sensors/Fans/<Fan-FQDD>
/redfish/v1/Chassis/System.Embedded.1/Sensors/Temperatures
/redfish/v1/Chassis/System.Embedded.1/Sensors/Temperatures/<Sensor-FQDD>
/redfish/v1/Chassis/System.Embedded.1/Power
/redfish/v1/Chassis/System.Embedded.1/Power/PowerControl
/redfish/v1/Chassis/System.Embedded.1/Sensors/Voltages
/redfish/v1/Chassis/System.Embedded.1/Sensors/Voltages/<Voltage-FQDD>
/redfish/v1/Chassis/System.Embedded.1/Power/PowerSupplies
/redfish/v1/Chassis/System.Embedded.1/Power/PowerSupplies/<PSU-FQDD>
/redfish/v1/Chassis/System.Embedded.1/Power/Redundancy/<PSRedundancy-FQDD>

More than just GET!

Example: Reboot Server

```
$ curl https://<OOB>/redfish/v1/Systems/System.Embedded.1/Actions/ComputerSystem.Reset \
     --request POST \
     --header "Content-Type: application/json" \
     --data '{"ResetType":"GracefulRestart"}' \
     --user root:password
```

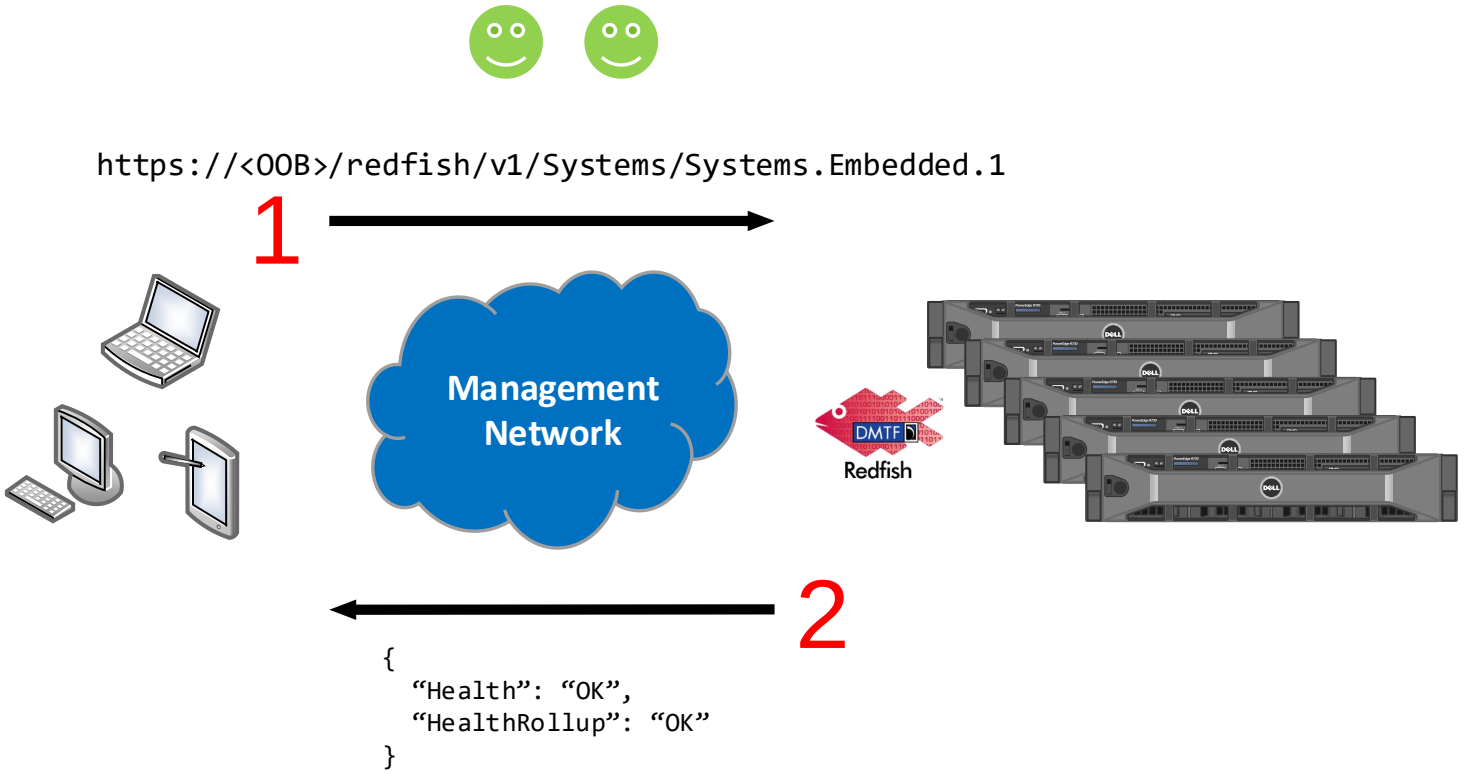
Example: Change boot mode to UEFI

```
$ curl https://<OOB>/redfish/v1/Systems/System.Embedded.1/Bios/Settings \
     --request PATCH \
     --header "Content-Type: application/json" \
     --data '{"Attributes":{"BootMode":"Uefi"}}' \
     --user root:password
```

Redfish Roadmap

- Version 1.x focused on servers. Will expand to cover storage and network infrastructure.
- Will expand APIs over time to cover new technologies such as NVDIMMs and Multifunction Network Adapters.
- SNIA is developing ‘Swordfish’ to address advanced storage devices.
- DMTF expanding open source efforts (http://github.com/dmtf)
  - Client libraries (Python, Java, PowerShell)
  - Redfish Mockup Creator / Server
  - Redfishtool (CLI utility similar to ipmitool)

Redfish provides scalability to OOB management

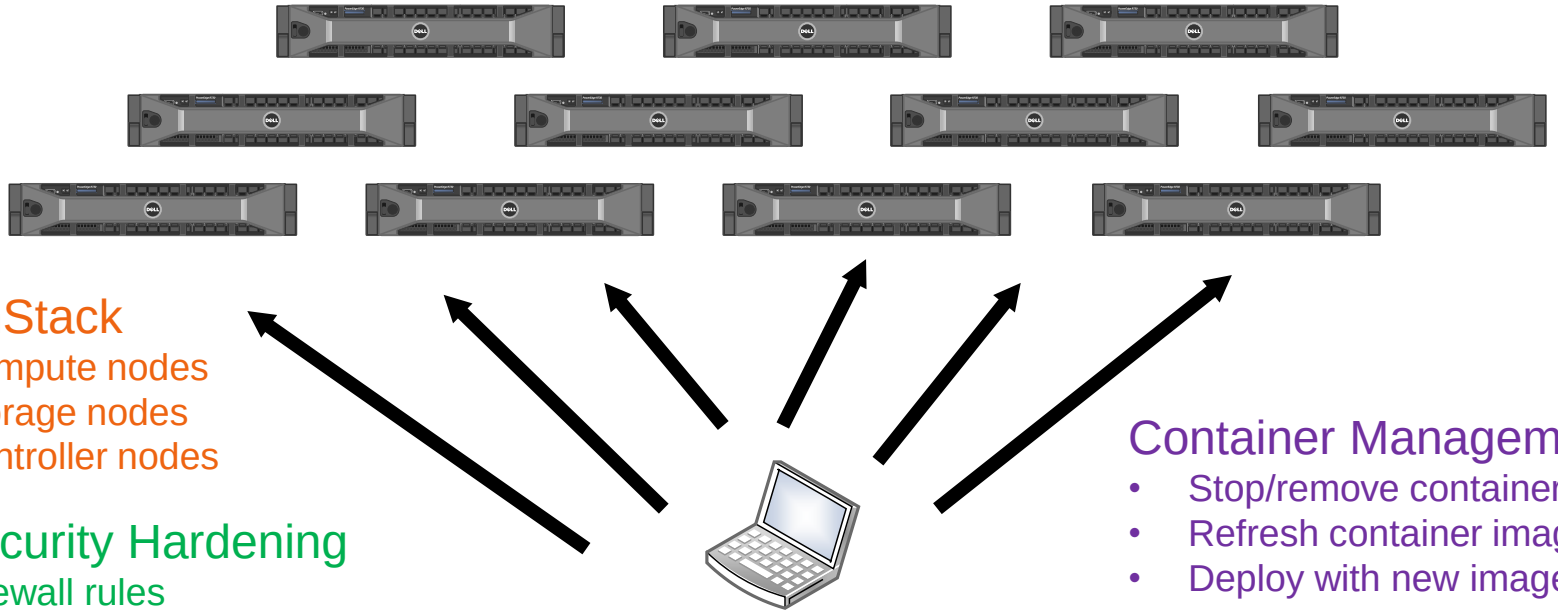


Ansible Overview

Ansible 101

- Automation software → makes repetitive tasks easy
- Agentless → minimum footprint
- No database backend → easy to install
- Remote tasks are run in parallel → fast & efficient
- Easier to learn and use than shell scripts

Ansible use cases



OpenStack

- Compute nodes
- Storage nodes
- Controller nodes

IT Security Hardening

- Firewall rules
- Remove unused login IDs
- Install updates

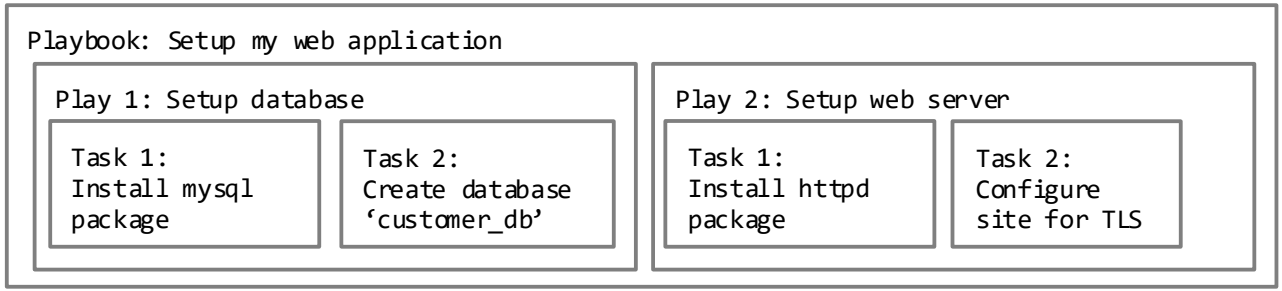
Container Management

- Stop/remove containers
- Refresh container images
- Deploy with new images

✓ 1-to-n management
✓ Executes tasks in parallel

Ansible concepts

- **Task:** A task is the smallest unit of work. Examples: “install a package”, “remove a user”, “create firewall rule” or “copy a file to this directory”.
- **Play:** A play is made up of tasks. Example: the play “*Prepare a database*” is composed of tasks:
  - ✓ Task 1: “Install the database package”
  - ✓ Task 2: “Set password”
  - ✓ Task 3: “Create database”
- **Playbook:** A playbook is composed of plays. Example: the playbook “*Setup my web application*” has plays 1) “Set up database server” and 2) “Set up web server”.



Simple implementation example

Say you provision 100 servers every day and you run these commands in each server:

```
$ groupadd admin
$ useradd -c “Sys Admin” -g admin -m sysman
$ mkdir /opt/tools
$ chmod 755 /opt/tools
$ chown sysman /opt/tools
$ yum -y install httpd
$ yum -y update
$ systemctl enable httpd
$ systemctl start httpd
$ rm /etc/motd
```

The same commands can be placed in an Ansible *playbook* and executed in 100 servers:

daily_tasks.yml

```
- name: daily tasks
  hosts: my_100_daily_servers
  tasks:
  - group: name=admin state=present
  - user: name=sysman comment="Sys Admin" group=admin
  - file: path=/opt/tools state=directory owner=sysman
    mode=0755
  - yum: name=httpd state=latest
  - yum: name=* state=latest
  - service: name=httpd state=started enabled=yes
  - file: path=/etc/motd state=absent
```

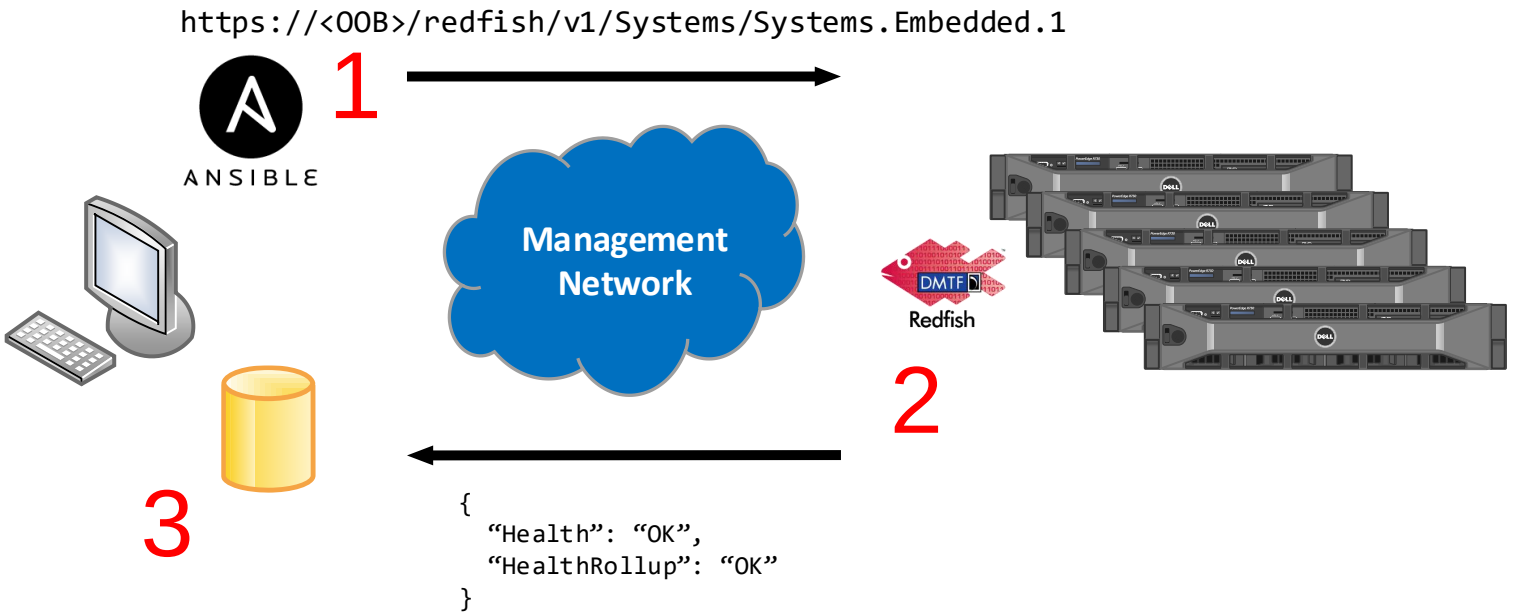
```
$ ansible-playbook daily_tasks.yml
```

Ansible module

- Whereas a playbook is where you specify the tasks to run; a module is the code to implement those tasks.
- Modules can be written in any language, but most popular is Python.
- If you are a system administrator, you will work mostly with playbooks.
- If you are a developer, you will work mostly with modules.

```
- name: daily tasks
  hosts: my_100_daily_servers
  tasks:
  - group: name=admin state=present
  - user: name=sysman comment="Sys Admin" group=admin
  - file: path=/opt/tools state=directory owner=sysman
    mode=0755
  - yum: name=httpd state=latest
  - yum: name=* state=latest
  - service: name=httpd state=started enabled=yes
  - file: path=/etc/motd state=absent
```

Scalable and automated OOB management



Coming together:
Ansible module for Redfish

DELL EMC

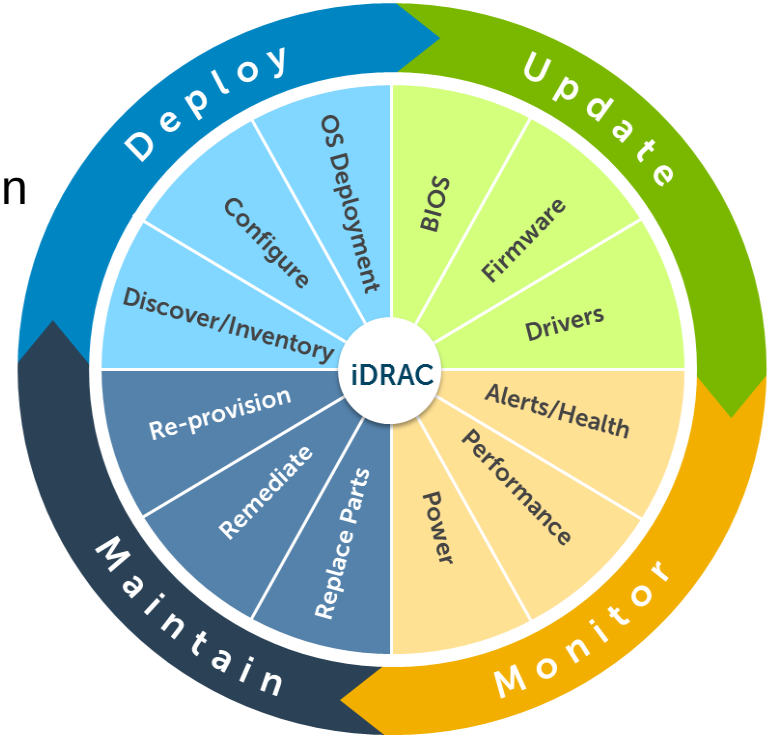
Ansible module for Redfish

- Use it to manage your entire IT infrastructure (compute, network & storage) from one controller.
- Automated inventory, monitoring & provisioning at scale.
- It’s open source, so you can write your own extensions and contribute back to the community.
- Working to submit upstream.
- DMTF will extend it and support it.

Key Lifecycle Management Tasks

- Device Inventory
- iDRAC Configuration

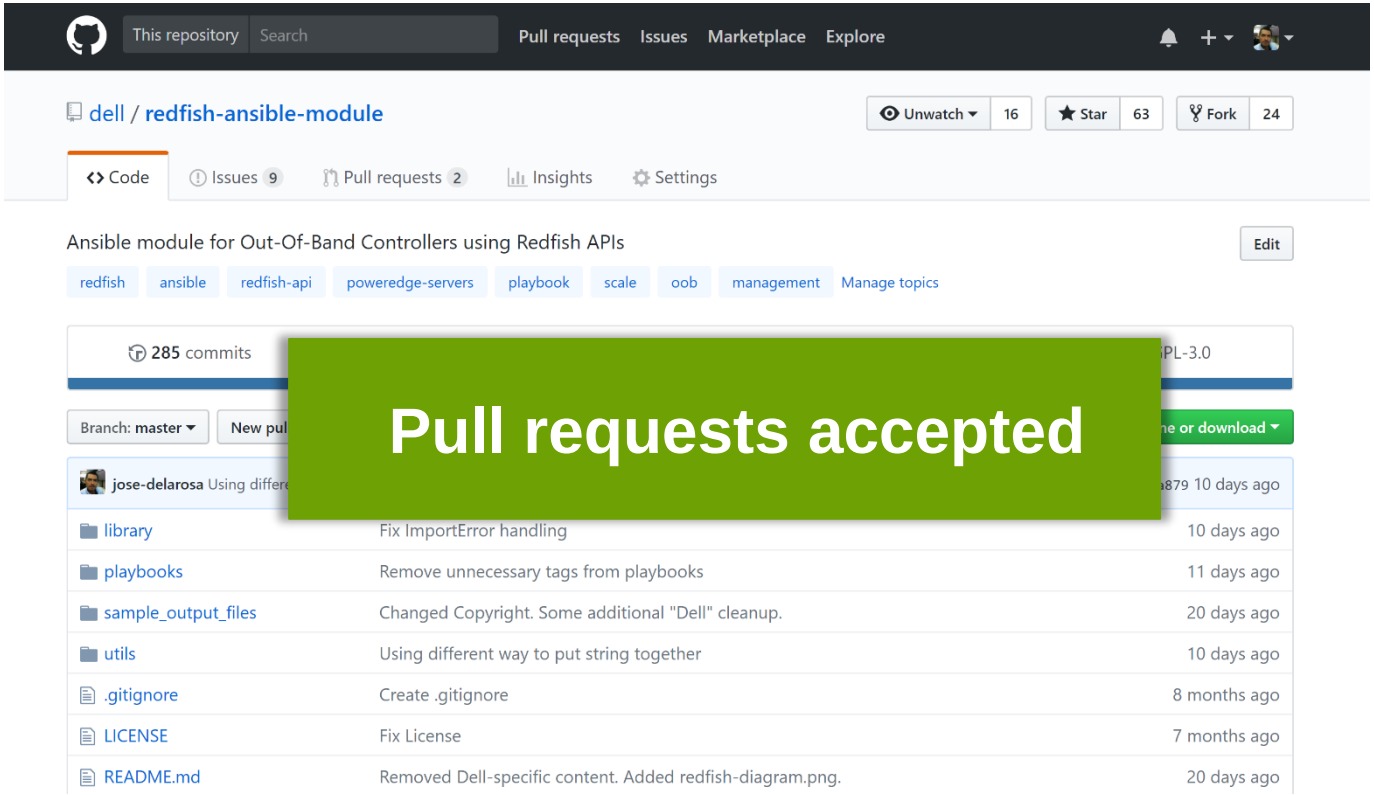
- Event Logs



- BIOS Configuration
- Firmware Update

- Health Reporting
- Power Management

https://github.com/dell/redfish-ansible-module



Example: get system inventory

1. Playbook

```
---
- hosts: myhosts
  name: System Inventory
  gather_facts: False

  tasks:

  - name: Define output file
    include_tasks: create_output_file.yml type=Syst

  - name: Getting system inventory
    local_action: >
      redfish category=Inventory command=GetSystem
      baseuri={{baseuri}} user={{user}} password={
    register: result

  - name: Copy results to output file
    local_action: copy content={{ result | to_nice_
      dest={{template}}.json
```

2. Execute Playbook

```
$ ansible-playbook system-inventory.yml

PLAY [PowerEdge iDRAC Get System Inventory]
************************************

TASK [Define timestamp]
*********************************************************
ok: [r740-1]
ok: [r630]
ok: [r640-1]

TASK [Define file to place results]
*****************************************
ok: [r630]
ok: [r640-1]
ok: [r740-1]

TASK [Create dropoff directory for host]
***************************************
changed: [r740-1 -> localhost]
changed: [r630 -> localhost]
changed: [r640-1 -> localhost]

TASK [Getting system inventory]
*********************************************
ok: [r740-1 -> localhost]
ok: [r640-1 -> localhost]
ok: [r630 -> localhost]

TASK [Copying results to file]
***********************************************
changed: [r630 -> localhost]
changed: [r740-1 -> localhost]
changed: [r640-1 -> localhost]

PLAY RECAP
*********************************************************************
r630                       : ok=5    changed=2    unreachable=0
r640-1                     : ok=5    changed=2    unreachable=0
r740-1                     : ok=5    changed=2    unreachable=0

Playbook run took 0 days, 0 hours, 0 minutes, 8 seconds
```

3. Result

r640-1_20171016_163922_inventory.json


```
{
    "changed": false,
    "result": {
        "AssetTag": "",
        "BiosVersion": "1.0.7",
        "BootSourceOverrideMode": "UEFI",
        "CpuCount": 2,
        "CpuHealth": "OK",
        "CpuModel": "Intel(R) Xeon(R) Silver 4108 CPU
        "HostName": "",
        "Manufacturer": "Dell Inc.",
        "MemoryHealth": "OK",
        "MemoryTotal": 128.0,
        "Model": "PowerEdge R640",
        "PartNumber": "008R9MA02",
        "PowerState": "On",
        "SerialNumber": "CNIV[redacted]0347",
        "ServiceTag": "3M[redacted]2",
        "Status": "OK",
        "SystemType": "Physical"
    }
}
```


DELL EMC

Example: Inventory spreadsheet

Server	iDRAC IP	Model	IP address	BIOS	CPU	Type	RAM	Service Tag	Status
webserver-1	192.168.2.10	PowerEdge R630	10.0.1.30	2.3.4	2	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz	128	5W14Q52	OK
webserver-2	192.168.2.11	PowerEdge R630	10.0.1.31	2.3.4	2	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz	128	5XXYQ32	OK
webserver-3	192.168.2.12	PowerEdge R630	10.0.1.33	2.3.2	2	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz	128	5XT3QYY	OK
appserver-1	192.168.2.13	PowerEdge R830	10.0.1.34	2.3.2	4	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.60GHz	512	5XR7QXY	OK
dbserver-1	192.168.3.10	PowerEdge R740	10.0.2.30	1.2.11	2	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.33GHz	256	5XR7Q88	OK
dbserver-2	192.168.3.11	PowerEdge R740	10.0.2.31	1.1.7	2	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.33GHz	256	5WEYQ37	OK
dbserver-3	192.168.3.12	PowerEdge R740	10.0.2.32	1.2.11	2	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.33GHz	256	5WR4Q12	Fail
dbserver-4	192.168.3.13	PowerEdge T640	10.0.2.33	1.1.3	2	Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.33GHz	512	5TEEQ21	OK
dbserver-5	192.168.3.14	PowerEdge T640	10.0.2.34	1.2.11	2	Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.33GHz	512	5TT1Q26	OK

Example: Set controller's NTP server

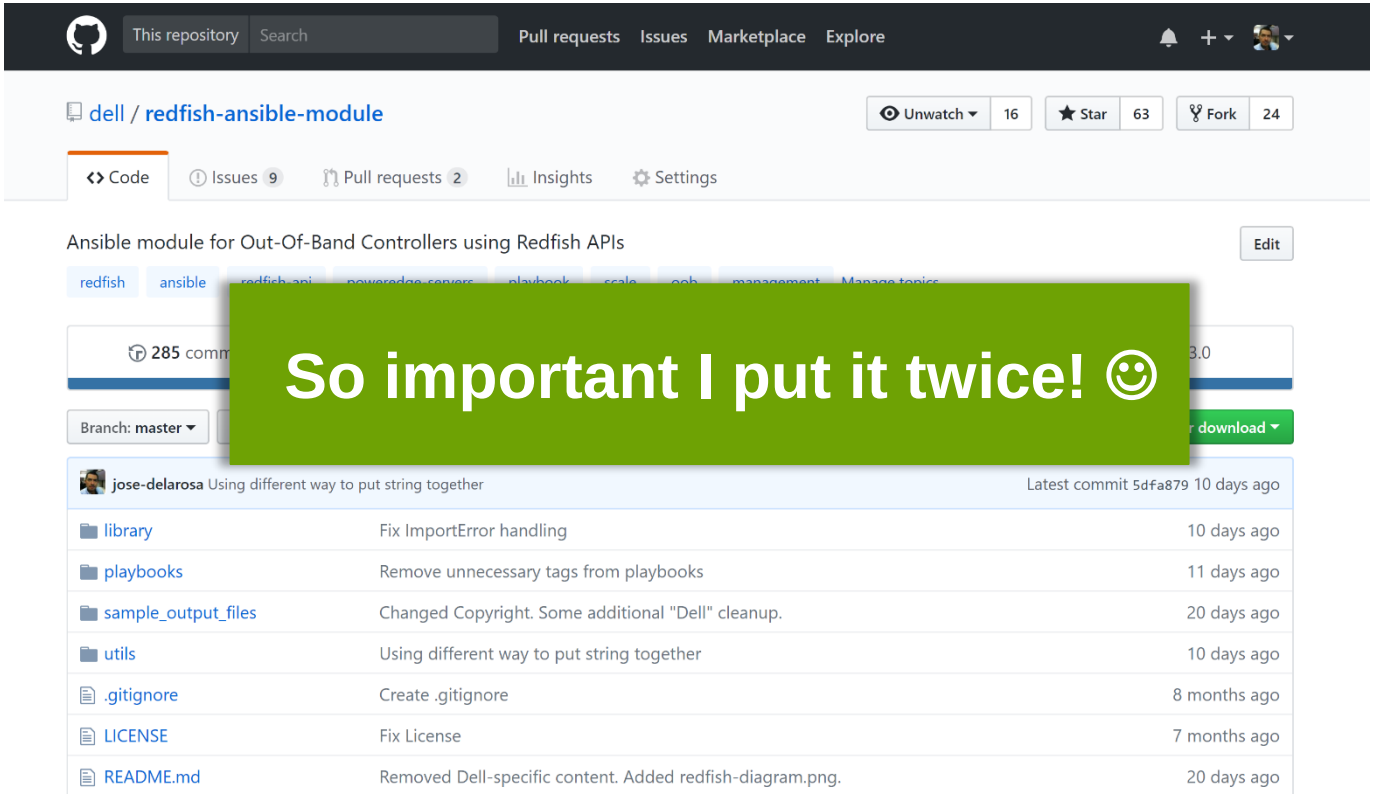
1. Playbook

```
---

- hosts: myhosts
  name: Set Manager NTP settings
  gather_facts: False

  vars:
    - ntpserver1: ntp.domain.com

  tasks:

  - name: Enable NTP
    local_action: >
       redfish category=Manager command=SetAttributes
       user={{user}} password={{password}} baseuri={{baseuri}}
       mgr_attr_name=NTPConfigGroup.1.NTPEnable mgr_attr_value=Enabled
    ignore_errors: yes

  - name: Set NTP server 1
    local_action: >
       redfish category=Manager command=SetAttributes
       user={{user}} password={{password}} baseuri={{baseuri}}
       mgr_attr_name=NTPConfigGroup.1.NTP1 mgr_attr_value={{ntpserver1}}
    ignore_errors: yes

  # Add more NTP servers as needed
  # To get exact attributes names, run the getattributes task first
```

2. Execute Playbook

```
$ ansible-playbook set_manager_ntp.yml

PLAY [Set Manager NTP settings] ************************************************

TASK [Enable NTP] **************************************************************
ok: [red1 -> localhost]
ok: [red4 -> localhost]
ok: [red2 -> localhost]
ok: [red3 -> localhost]
fatal: [t620 -> localhost]: FAILED! => {"changed": false, "msg": "Resource not supported"}
...ignoring

TASK [Set NTP server 1] ********************************************************
ok: [red1 -> localhost]
ok: [red4 -> localhost]
ok: [red2 -> localhost]
ok: [red3 -> localhost]
fatal: [t620 -> localhost]: FAILED! => {"changed": false, "msg": "Resource not supported"}
...ignoring

PLAY RECAP *********************************************************************
red1                       : ok=2    changed=0    unreachable=0    failed=0
red2                       : ok=2    changed=0    unreachable=0    failed=0
red3                       : ok=2    changed=0    unreachable=0    failed=0
red4                       : ok=2    changed=0    unreachable=0    failed=0
t620                       : ok=2    changed=0    unreachable=0    failed=0

Playbook run took 0 days, 0 hours, 0 minutes, 37 seconds
```

https://github.com/dell/redfish-ansible-module



Resources

- Redfish API specification: http://bit.ly/2gb9VBj
- Getting started with Ansible: http://bit.ly/2oCj5xy
- PowerEdge Redfish API Overview: http://dell.to/2odsH1p
- iDRAC Redfish API Reference Guide: http://dell.to/2oyjMTy
- jq JSON parser: https://stedolan.github.io/jq/

Conclusion

- Automation + scalability are useful when managing hardware.
- Module tested mostly on Dell EMC platforms, but *should* work on any controller that implements the Redfish API standard.
- Module was designed to be as simple as possible.
- Thank you for listening, hope this was useful.

Thank you

Q & A


DELL EMC