Sushy Tools Documentation

Release 1.3.1.dev15

OpenStack Foundation

Feb 19, 2025

CONTENTS

This is a set of simple simulation tools aimed at supporting the development and testing of the Redfish protocol implementations and, in particular, the Sushy library (https://docs.openstack.org/sushy/). It is not designed for use outside of development and testing environments. Please do not run sushy-tools in a production environment of any kind.

The package ships two simulators - the static Redfish responder and the virtual Redfish BMC (which is backed by libvirt or OpenStack cloud).

The static Redfish responder is a simple REST API server which responds with the same things to client queries. It is effectively read-only.

The virtual Redfish BMC resembles the real Redfish-controlled bare metal machine to some extent. Some client queries are translated to commands that actually control VM instances simulating bare metal hardware. However, some of the Redfish commands just return static content, never touching the virtualization backend and in this regard, the virtual Redfish BMC is similar to the static Redfish responder.

- Free software: Apache license
- Documentation: https://docs.openstack.org/sushy-tools
- Source: http://opendev.org/openstack/sushy-tools
- Bugs: https://storyboard.openstack.org/#!/project/openstack/sushy-tools

CHAPTER

ONE

DOCUMENTATION

1.1 Installation

The sushy-tools Python package can be downloaded and installed with *pip*:

```
$ pip install sushy-tools
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv sushy-tools
$ pip install sushy-tools
```

The *Virtual Redfish BMC* tool relies upon one or more hypervisors to mimic bare metal nodes. Depending on the virtualization backend you are planning to use, certain third-party dependencies should also be installed.

The dependencies for the virtualization backends that should be installed for the corresponding drivers to become operational are:

- *libvirt-python* for the libvirt driver
- *openstacksdk* for the nova driver

Note

The dependencies for at least one virtualization backend should be satisfied to have the *Virtual Redfish BMC* emulator operational.

1.2 Configuring emulators

1.2.1 Running emulators in the background

The emulators run as interactive processes attached to the terminal by default. You can create systemd services to run the emulators in the background. For each emulator, create a systemd unit file, and update `<full-path>` to the `sushy-static` or `sushy-emulator` binary, and adjust the arguments as necessary, for example:

```
[Unit]
Description=Sushy Libvirt emulator
After=syslog.target

```

(continues on next page)

(continued from previous page)

```
[Service]
Type=simple
ExecStart=/<full-path>/sushy-emulator --port 8000 --libvirt-uri "qemu:///
↪system"
StandardOutput=syslog
StandardError=syslog
```

If you want to run the emulators with different configurations, for example, the `sushy-static` emulator with different mockup files, then create a new systemd unit file.

You can also use `gunicorn` to run `sushy-emulator`, for example:

```
ExecStart=/usr/bin/gunicorn sushy_tools.emulator.main:app
```

1.2.2 Using configuration file

Besides command-line options, *sushy-emulator* can be configured via a configuration file. This tool uses the Flask application configuration infrastructure. Emulator-specific configuration options are prefixed with **SUSHY_EMULATOR_** to make sure that they don't collide with Flask's own configuration options.

The configuration file itself can be specified through the *SUSHY_EMULATOR_CONFIG* environment variable.

The full list of supported options and their meanings can be found in the sample configuration file:

```
# sushy emulator configuration file built on top of Flask application
# configuration infrastructure: http://flask.pocoo.org/docs/config/

# Listen on all local IP interfaces
SUSHY_EMULATOR_LISTEN_IP = u''

# Bind to TCP port 8000
SUSHY_EMULATOR_LISTEN_PORT = 8000

# Serve this SSL certificate to the clients
SUSHY_EMULATOR_SSL_CERT = None

# If SSL certificate is being served, this is its RSA private key
SUSHY_EMULATOR_SSL_KEY = None

# If authentication is desired, set this to an htpasswd file.
SUSHY_EMULATOR_AUTH_FILE = None

# The OpenStack cloud ID to use. This option enables OpenStack driver.
SUSHY_EMULATOR_OS_CLOUD = None

# If image should created via file upload instead of web-download based␣
↪image
# import OpenStack cloud virtual media
SUSHY_EMULATOR_OS_VMEDIA_IMAGE_FILE_UPLOAD = False
```

(continues on next page)

(continued from previous page)

```
# The OpenStack cloud ID to use for Ironic. This option enables Ironic␣
↪driver.
SUSHY_EMULATOR_IRONIC_CLOUD = None

# The libvirt URI to use. This option enables libvirt driver.
SUSHY_EMULATOR_LIBVIRT_URI = u'qemu:///system'

# Instruct the libvirt driver to ignore any instructions to set the boot␣
↪device,
# allowing the UEFI firmware to instead rely on the EFI Boot Manager.
# Note: This sets the legacy boot element to dev="fd" and relies on the␣
↪floppy
# not existing. It likely won't work if your VM has a floppy drive.
SUSHY_EMULATOR_IGNORE_BOOT_DEVICE = False


# The map of firmware loaders dependent on the boot mode and system
# architecture. Ideally the x86_64 loader will be capable of secure boot or␣
↪not
# based on the chosen nvram.
SUSHY_EMULATOR_BOOT_LOADER_MAP = {
    'UEFI': {
        'x86_64': u'/usr/share/OVMF/OVMF_CODE.secboot.fd',
        'aarch64': u'/usr/share/AAVMF/AAVMF_CODE.fd'
    },
    'Legacy': {
        'x86_64': None,
        'aarch64': None
    }
}

# nvram templates to use on x86_64 to enable or disable secure boot
SUSHY_EMULATOR_SECURE_BOOT_ENABLED_NVRAM = '/usr/share/OVMF/OVMF_VARS.
↪secboot.fd'
SUSHY_EMULATOR_SECURE_BOOT_DISABLED_NVRAM = '/usr/share/OVMF/OVMF_VARS.fd'

# This map contains statically configured Redfish Chassis linked up with the
# Systems and Managers enclosed into this Chassis.
#
# The first chassis in the list will contain all other resources.
#
# If this map is not present in the configuration, a single default Chassis␣
↪is
# configured automatically to enclose all available Systems and Managers.
SUSHY_EMULATOR_CHASSIS = [
    {
        u'Id': u'Chassis',
        u'Name': u'Chassis',
        u'UUID': u'48295861-2522-3561-6729-621118518810'
```

(continues on next page)

(continued from previous page)

```
    }
]

# This map contains statically configured Redfish IndicatorLED resource␣
↪state
# ('Lit', 'Off', 'Blinking'), keyed by UUIDs of System and Chassis␣
↪resources.
#
# If this map is not present in the configuration, each System and Chassis␣
↪will
# have their IndicatorLED `Lit` by default.
#
# The Redfish client can change IndicatorLED state. The new state is␣
↪volatile,
# i.e. it's maintained in process memory.
SUSHY_EMULATOR_INDICATOR_LEDS = {
#    u'48295861-2522-3561-6729-621118518810': u'Blinking'
}

# This map contains statically configured virtual media resources.
# These devices ('Cd', 'Floppy', 'USBStick') will be exposed by the␣
↪Manager(s)
# and possibly used by the System(s) if system emulation backend supports␣
↪boot
# image configuration.
#
# This value is ignored by the OpenStack driver, which only supports the 'Cd
↪'
# device. If this map is not present in the configuration, the following
# configuration is used for other drivers:
SUSHY_EMULATOR_VMEDIA_DEVICES = {
    u'Cd': {
        u'Name': 'Virtual CD',
        u'MediaTypes': [
            u'CD',
            u'DVD'
        ]
    },
    u'Floppy': {
        u'Name': u'Virtual Removable Media',
        u'MediaTypes': [
            u'Floppy',
            u'USBStick'
        ]
    }
}

# Instruct the virtual media insertion to not verify the SSL certificate␣
↪when
```

(continues on next page)

(continued from previous page)

```
# retrieving the image.
SUSHY_EMULATOR_VMEDIA_VERIFY_SSL = False

# This map contains statically configured Redfish Storage resources linked␣
↪up
# with the Systems resources, keyed by the UUIDs of the Systems.
SUSHY_EMULATOR_STORAGE = {
    "da69abcc-dae0-4913-9a7b-d344043097c0": [
        {
            "Id": "1",
            "Name": "Local Storage Controller",
            "StorageControllers": [
                {
                    "MemberId": "0",
                    "Name": "Contoso Integrated RAID",
                    "SpeedGbps": 12
                }
            ],
            "Drives": [
                "32ADF365C6C1B7BD"
            ]
        }
    ]
}

# This map contains statically configured Redfish Drives resources. The␣
↪Drive
# objects are keyed in a composite fashion using a tuple of the form
# (System_UUID, Storage_ID) referring to the UUID of the System and Id of␣
↪the
# Storage resource, respectively, to which the Drive belongs.
SUSHY_EMULATOR_DRIVES = {
    ("da69abcc-dae0-4913-9a7b-d344043097c0", "1"): [
        {
            "Id": "32ADF365C6C1B7BD",
            "Name": "Drive Sample",
            "CapacityBytes": 899527000000,
            "Protocol": "SAS"
        }
    ]
}

# This map contains dynamically configured Redfish Volume resources backed␣
↪by
# the libvirt virtualization backend of the dynamic Redfish emulator.
# The Volume objects are keyed in a composite fashion using a tuple of the␣
↪form
# (System_UUID, Storage_ID) referring to the UUID of the System and ID of␣
↪the
```

(continues on next page)

(continued from previous page)

```
# Storage resource, respectively, to which the Volume belongs.
#
# Only the Volumes specified in the map or created via a POST request are
# allowed to be emulated upon by the emulator. Volumes other than these can
# neither be listed nor deleted.
#
# The Volumes in the map missing from the libvirt backend will be created
# dynamically in the pool name specified (provided the pool exists in the
# backend). If the pool name is not specified, the Volume will be created
# automatically in a pool named 'default'.
SUSHY_EMULATOR_VOLUMES = {
    ('da69abcc-dae0-4913-9a7b-d344043097c0', '1'): [
        {
            "libvirtPoolName": "sushyPool",
            "libvirtVolName": "testVol",
            "Id": "1",
            "Name": "Sample Volume 1",
            "VolumeType": "Mirrored",
            "CapacityBytes": 23748
        },
        {
            "libvirtPoolName": "sushyPool",
            "libvirtVolName": "testVol1",
            "Id": "2",
            "Name": "Sample Volume 2",
            "VolumeType": "StripedWithParity",
            "CapacityBytes": 48395
        }
    ]
}

# This list contains the identities of instances that the driver will␣
↪filter by.
# It is useful in a tenant environment where only some instances represent
# virtual bare metal.
SUSHY_EMULATOR_ALLOWED_INSTANCES = [
    "437XR1138R2",
    "1",
    "529QB9450R6",
    "529QB9451R6",
    "529QB9452R6",
    "529QB9453R6"
]

# Disable the ability to power off the node, in line with NCSI enablement in
# Ironic
SUSHY_EMULATOR_DISABLE_POWER_OFF = False
```

1.3 Using Redfish emulators

The sushy-tools package includes two emulators - static and dynamic.

The static emulator can be used to serve Redfish mocks in the form of static JSON documents. The dynamic emulator relies upon the *libvirt*, *OpenStack* or *Ironic* virtualization backends to mimic nodes behind a Redfish BMC.

1.3.1 Static Redfish BMC

The static Redfish responder is a simple REST API server which serves static contents down to the Redfish client. The tool emulates the simple read-only BMC.

The user is expected to supply the Redfish-compliant contents, perhaps downloaded from the DMTF web site. For example, this .zip archive includes Redfish content mocks for Redfish 1.0.0.

```
curl -o DSP2043_1.0.0.zip \
     https://www.dmtf.org/sites/default/files/standards/documents/DSP2043_1.
↪0.0.zip
unzip -d mockups DSP2043_1.0.0.zip
sushy-static -m mockups/public-rackmount
```

Once you have the static emulator running, you can use it as if it was a read-only bare metal controller listening at *localhost:8000* (by default):

```
curl http://localhost:8000/redfish/v1/Systems/
{
    "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
    "Name": "Computer System Collection",
    "Members@odata.count": 1,
    "Members": [
        {
            "@odata.id": "/redfish/v1/Systems/437XR1138R2"
        }
    ],
    "@odata.context": "/redfish/v1/$metadata#Systems",
    "@odata.id": "/redfish/v1/Systems",
    "@Redfish.Copyright": "Copyright 2014-2016 Distributed Management Task␣
↪Force, Inc. (DMTF). For the full DMTF copyright policy, see http://www.
↪dmtf.org/about/policies/copyright."
}
```

You can mock different Redfish versions as well as different bare metal machines by providing the appropriate Redfish contents.

1.3.2 Virtual Redfish BMC

The Virtual Redfish BMC emulator is functionally similar to the Virtual BMC tool, except that the frontend protocol is Redfish rather than IPMI. The Redfish commands coming from the client are handled by one or more resource-specific drivers.

Feature sets

The emulator can be configured with different feature sets to emulate different hardware. The feature set is supplied either via the `SUSHY_EMULATOR_FEATURE_SET` configuration variable or through the `--feature-set` command line flag.

Supported feature sets are: * `minimum` - only Systems with Boot settings and no other optional fields. * `vmedia` - `minimum` plus Managers, VirtualMedia and EthernetInterfaces. * `full` - all features implemented in the emulator.

Systems resource

For the *Systems* resource, the emulator maintains two drivers relying on a virtualization backend to emulate bare metal machines by means of virtual machines. In addition, there is a fake driver used to mock bare metal machines.

The following sections will explain how to configure and use each of these drivers.

Systems resource driver: libvirt

The first thing you need is to set up some libvirt-managed virtual machines (AKA domains) to manipulate. The following command will create a new virtual machine i.e. libvirt domain *vbmc-node*:

```
tmpfile=$(mktemp /tmp/sushy-domain.XXXXXX)
virt-install \
   --name vbmc-node \
   --ram 1024 \
   --disk size=1 \
   --vcpus 2 \
   --os-type linux \
   --os-variant fedora28 \
   --graphics vnc \
   --print-xml > $tmpfile
virsh define --file $tmpfile
rm $tmpfile
```

Next you can fire up the Redfish virtual BMC which will listen at *localhost:8000* (by default):

```
sushy-emulator
 * Running on http://localhost:8000/ (Press CTRL+C to quit)
```

Now you should be able to see your libvirt domain among the Redfish *Systems*:

```
curl http://localhost:8000/redfish/v1/Systems/
{
    "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
    "Name": "Computer System Collection",
    "Members@odata.count": 1,
    "Members": [

            {
                "@odata.id": "/redfish/v1/Systems/vbmc-node"
            }
```

(continues on next page)

(continued from previous page)

```
    ],
    "@odata.context": "/redfish/v1/$metadata#ComputerSystemCollection.
↪ComputerSystemCollection",
    "@odata.id": "/redfish/v1/Systems",
    "@Redfish.Copyright": "Copyright 2014-2016 Distributed Management Task␣
↪Force, Inc. (DMTF). For the full DMTF copyright policy, see http://www.
↪dmtf.org/about/policies/copyright."
}
```

You should be able to flip its power state via the Redfish call:

```
curl -d '{"ResetType":"On"}' \
    -H "Content-Type: application/json" -X POST \
     http://localhost:8000/redfish/v1/Systems/vbmc-node/Actions/
↪ComputerSystem.Reset

curl -d '{"ResetType":"ForceOff"}' \
    -H "Content-Type: application/json" -X POST \
     http://localhost:8000/redfish/v1/Systems/vbmc-node/Actions/
↪ComputerSystem.Reset
```

You can have as many domains as you need. The domains can be concurrently managed over Redfish and some other tool like *Virtual BMC*.

Simple Storage resource

For emulating the *Simple Storage* resource, some additional preparation is required on the host side.

First, you need to create, build and start a libvirt storage pool using virsh:

```
virsh pool-define-as testPool dir - - - - "/testPool"
virsh pool-build testPool
virsh pool-start testPool
virsh pool-autostart testPool
```

Next, create a storage volume in the above created storage pool:

```
virsh vol-create-as testPool testVol 1G
```

Next, attach the created volume to the virtual machine/domain:

```
virsh attach-disk vbmc-node /testPool/testVol sda
```

Now, query the *Simple Storage* resource collection for the *vbmc-node* domain in a closely similar format (with 'ide' and 'scsi', here, referring to the two Redfish Simple Storage Controllers available for this domain):

```
curl http://localhost:8000/redfish/v1/vbmc-node/SimpleStorage
{
    "@odata.type": "#SimpleStorageCollection.SimpleStorageCollection",
    "Name": "Simple Storage Collection",
```

(continues on next page)

(continued from previous page)

```
    "Members@odata.count": 2,
    "Members": [

            {
                "@odata.id": "/redfish/v1/Systems/vbmc-node/
↪SimpleStorage/ide"
            },

            {
                "@odata.id": "/redfish/v1/Systems/vbmc-node/
↪SimpleStorage/scsi"
            }

    ],
    "Oem": {},
    "@odata.context": "/redfish/v1/$metadata#SimpleStorageCollection.
↪SimpleStorageCollection",
    "@odata.id": "/redfish/v1/Systems/vbmc-node/SimpleStorage"
}
```

UEFI boot

By default, *legacy* or *BIOS* mode is used to boot the instance. However, the libvirt domain can be configured to boot via UEFI firmware. This process requires additional preparation on the host side.

On the host you need to have OVMF firmware binaries installed. Fedora users could pull them as *edk2-ovmf* RPM. On Ubuntu, *apt-get install ovmf* should do the job.

Then you need to create a VM by running *virt-install* with the UEFI-specific *–boot* options:

Example:

```
tmpfile=$(mktemp /tmp/sushy-domain.XXXXXX)
virt-install \
   --name vbmc-node \
   --ram 1024 \
   --boot loader.readonly=yes \
   --boot loader.type=pflash \
   --boot loader.secure=no \
   --boot loader=/usr/share/OVMF/OVMF_CODE.secboot.fd \
   --boot nvram.template=/usr/share/OVMF/OVMF_VARS.fd \
   --disk size=1 \
   --vcpus 2 \
   --os-type linux \
   --os-variant fedora28 \
   --graphics vnc \
   --print-xml > $tmpfile
virsh define --file $tmpfile
rm $tmpfile
```

This will create a new *libvirt* domain with the path to OVMF images properly configured. Let's take a note on the path to the blob by running *virsh dumpxml vbmc-node*:

Example:

```
<domain type="kvm">
  ...
  <os>
    <type arch="x86_64" machine="q35">hvm</type>
    <loader readonly="yes" type="pflash" secure="no">/usr/share/edk2/ovmf/
↪OVMF_CODE.secboot.fd</loader>
    <nvram template="/usr/share/edk2/ovmf/OVMF_VARS.fd"/>
    <boot dev="hd"/>
  </os>
  ...
</domain>
```

Because now we need to add this path to the emulator's configuration matching the VM architecture we are running. It is also possible to make Redfish calls to enable or disable Secure Boot by specifying which nvram template to load in each case. Make a copy of the stock configuration file and edit it accordingly:

```
$ cat sushy-tools/doc/source/admin/emulator.conf
...
SUSHY_EMULATOR_BOOT_LOADER_MAP = {
    'Uefi': {
        'x86_64': '/usr/share/OVMF/OVMF_CODE.secboot.fd',
        ...
}
SUSHY_EMULATOR_SECURE_BOOT_ENABLED_NVRAM = '/usr/share/OVMF/OVMF_VARS.
↪secboot.fd'
SUSHY_EMULATOR_SECURE_BOOT_DISABLED_NVRAM = '/usr/share/OVMF/OVMF_VARS.fd'
...
```

Now you can run *sushy-emulator* with the updated configuration file:

```
sushy-emulator --config emulator.conf
```

Note

The images you will serve to your VMs need to be UEFI-bootable.

Settable boot image

The *libvirt* system emulation backend supports setting custom boot images, so that libvirt domains (representing bare metal nodes) can boot from user images.

This feature enables system boot from virtual media device.

The limitations:

- Only ISO images are supported

See *VirtualMedia* resource section for more information on how to perform virtual media boot.

Systems resource driver: OpenStack

You can use OpenStack cloud instances to simulate Redfish-managed bare metal machines. This setup is known under the name of OpenStack Virtual Baremetal. We will largely reuse its OpenStack infrastructure and configuration instructions. After all, what we are trying to do here is to set up the Redfish emulator alongside the openstackbmc tool which is used for exactly the same purpose at OVB with the only difference being that it works over the *IPMI* protocol as opposed to *Redfish*.

The easiest way is probably to set up your OpenStack Virtual Baremetal cloud by following its instructions.

Once your OVB cloud is operational, you log into the *BMC* instance and *set up sushy-tools* there.

Next you can invoke the Redfish virtual BMC pointing it to your OVB cloud:

```
sushy-emulator --os-cloud rdo-cloud
 * Running on http://localhost:8000/ (Press CTRL+C to quit)
```

By this point you should be able to see your OpenStack instances among the Redfish *Systems*:

```
curl http://localhost:8000/redfish/v1/Systems/
{
    "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
    "Name": "Computer System Collection",
    "Members@odata.count": 1,
    "Members": [

            {
                "@odata.id": "/redfish/v1/Systems/8dbe91da-4002-4d61-a56d-
↪1a00fc61c35d"
            }

    ],
    "@odata.context": "/redfish/v1/$metadata#ComputerSystemCollection.
↪ComputerSystemCollection",
    "@odata.id": "/redfish/v1/Systems",
    "@Redfish.Copyright": "Copyright 2014-2016 Distributed Management Task␣
↪Force, Inc. (DMTF). For the full DMTF copyright policy, see http://www.
↪dmtf.org/about/policies/copyright."
}
```

And flip an instance's power state via the Redfish call:

```
curl -d '{"ResetType":"On"}' \
    -H "Content-Type: application/json" -X POST \
     http://localhost:8000/redfish/v1/Systems/vbmc-node/Actions/
↪ComputerSystem.Reset

curl -d '{"ResetType":"ForceOff"}' \
    -H "Content-Type: application/json" -X POST \
     http://localhost:8000/redfish/v1/Systems/vbmc-node/Actions/
↪ComputerSystem.Reset
```

You can have as many OpenStack instances as you need. The instances can be concurrently managed over Redfish and functionally similar tools.

Systems resource driver: Ironic

You can use the Ironic driver to manage Ironic baremetal instance to simulated Redfish API. You may want to do this if you require a redfish-compatible endpoint but don't have direct access to the BMC (you only have access via Ironic) or the BMC doesn't support redfish.

Assuming your bare metal cloud is set up you can invoke the Redfish emulator by running:

```
sushy-emulator --ironic-cloud baremetal-cloud
 * Running on http://localhost:8000/ (Press CTRL+C to quit)
```

By this point you should be able to see your Bare metal instances among the Redfish *Systems*:

```
curl http://localhost:8000/redfish/v1/Systems/
{
    "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
    "Name": "Computer System Collection",
    "Members@odata.count": 1,
    "Members": [

            {
                "@odata.id": "/redfish/v1/Systems/<uuid>"
            }

    ],
    "@odata.context": "/redfish/v1/$metadata#ComputerSystemCollection.
↪ComputerSystemCollection",
    "@odata.id": "/redfish/v1/Systems",
    "@Redfish.Copyright": "Copyright 2014-2016 Distributed Management Task␣
↪Force, Inc. (DMTF). For the full DMTF copyright policy, see http://www.
↪dmtf.org/about/policies/copyright."
}
```

And flip an instance's power state via the Redfish call:

```
curl -d '{"ResetType":"On"}' \
    -H "Content-Type: application/json" -X POST \
     http://localhost:8000/redfish/v1/Systems/<uuid>/Actions/ComputerSystem.
↪Reset

curl -d '{"ResetType":"ForceOff"}' \
    -H "Content-Type: application/json" -X POST \
     http://localhost:8000/redfish/v1/Systems/<uuid>/Actions/ComputerSystem.
↪Reset
```

Or update their boot device:

```
curl -d '{"Boot":{"BootSourceOverrideTarget":"Pxe"}}' \
    -H "Content-Type: application/json" -X PATCH \
     http://localhost:8000/redfish/v1/Systems/<uuid>

curl -d '{"Boot":{"BootSourceOverrideTarget":"Hdd"}}' \
```

(continues on next page)

(continued from previous page)

```
    -H "Content-Type: application/json" -X PATCH \
     http://localhost:8000/redfish/v1/Systems/<uuid>
```

Systems resource driver: fake

The `fake` system driver is designed to conduct large-scale testing of Ironic without having a lot of bare metal machines or being able to create a large number of virtual machines. When the Redfish emulator is configured with the `fake` system backend, all operations just return success. Any modifications are done purely in the local cache. This way, many Ironic operations can be tested at scale without access to a large computing pool.

System status notifications

The `fake` driver may need to simulate components that run on the VMs to test an end-to-end deployment. This requires a hook interface to integrate external components. For instance, when testing Ironic scalability, Ironic needs to communicate with the Ironic Python Agent (IPA). A fake IPA can be implemented and synchronized with the VM status using this hook, which notifies the fake IPA whenever the VM status changes.

To enable notifications, set `external_notifier` to `True` in the fake System object:

```
{
    "uuid": "7946b59-9e44-4fa7-8e91-f3527a1ef094",
    "name": "fake",
    "power_state": "Off",
    "external_notifier": True,
    "nics": [
        {
            "mac": "00:5c:52:31:3a:9c",
            "ip": "172.22.0.100"
        }
    ]
}
```

After this, whenever the fake driver updates this System object, it will send an HTTP `PUT` request with the new system object as `JSON` data. The endpoint URL can be configured with the parameter `EXTERNAL_NOTIFICATION_URL`.

Filtering by allowed instances

It is not always desirable to manage every accessible virtual machine as a Redfish System, such as when an OpenStack tenant has many instances which do not represent virtual bare metal. In this case it is possible to specify a list of UUIDs which are allowed.

```
$ cat sushy-tools/doc/source/admin/emulator.conf
...
SUSHY_EMULATOR_ALLOWED_INSTANCES = [
    "437XR1138R2",
    "1",
    "529QB9450R6",
```

(continues on next page)

(continued from previous page)

```
    "529QB9451R6",
    "529QB9452R6",
    "529QB9453R6"
]
...
```

Managers resource

Managers are emulated based on Systems: each *System* has a *Manager* with the same UUID. The first manager (alphabetically) will pretend to manage all *Chassis* and potentially other resources.

Managers will be revealed when querying the *Managers* resource directly, as well as other resources they manage or have some other relations.

```
curl http://localhost:8000/redfish/v1/Managers
{
    "@odata.type": "#ManagerCollection.ManagerCollection",
    "Name": "Manager Collection",
    "Members@odata.count": 1,
    "Members": [

            {
                "@odata.id": "/redfish/v1/Managers/58893887-8974-2487-2389-
↪841168418919"
            }

    ],
    "@odata.context": "/redfish/v1/$metadata#ManagerCollection.
↪ManagerCollection",
    "@odata.id": "/redfish/v1/Managers",
    "@Redfish.Copyright": "Copyright 2014-2017 Distributed Management Task␣
↪Force, Inc. (DMTF). For the full DMTF copyright policy, see http://www.
↪dmtf.org/about/policies/copyright."
```

Chassis resource

For emulating the *Chassis* resource, the user can statically configure one or more imaginary chassis. All existing resources (e.g. *Systems*, *Managers*, *Drives*) will pretend to reside in the first chassis.

```
SUSHY_EMULATOR_CHASSIS = [
    {
        "Id": "Chassis",
        "Name": "Chassis",
        "UUID": "48295861-2522-3561-6729-621118518810"
    }
]
```

By default a single chassis with be configured automatically.

Chassis will be revealed when querying the *Chassis* resource directly, as well as other resources they manage or have some other relations.

```
curl http://localhost:8000/redfish/v1/Chassis
{
    "@odata.type": "#ChassisCollection.ChassisCollection",
    "Name": "Chassis Collection",
    "Members@odata.count": 1,
    "Members": [
            {
                "@odata.id": "/redfish/v1/Chassis/48295861-2522-3561-6729-
↪621118518810"
            }
    ],
    "@odata.context": "/redfish/v1/$metadata#ChassisCollection.
↪ChassisCollection",
    "@odata.id": "/redfish/v1/Chassis",
    "@Redfish.Copyright": "Copyright 2014-2017 Distributed Management Task␣
↪Force, Inc. (DMTF). For the full DMTF copyright policy, see http://www.
↪dmtf.org/about/policies/copyright."
```

Indicator resource

The *IndicatorLED* resource is emulated as a persistent emulator database record, observable and manageable by a Redfish client.

By default, the *Chassis* and *Systems* resources have emulated *IndicatorLED* sub-resources attached and *Lit*.

Non-default initial indicator state can optionally be configured on a per-resource basis:

```
SUSHY_EMULATOR_INDICATOR_LEDS = {
    "48295861-2522-3561-6729-621118518810": "Blinking"
}
```

Indicator LEDs will be revealed when querying any resource having *IndicatorLED*:

```
$ curl http://localhost:8000/redfish/v1/Chassis/48295861-2522-3561-6729-
↪621118518810
{
    "@odata.type": "#Chassis.v1_5_0.Chassis",
    "Id": "48295861-2522-3561-6729-621118518810",
    "Name": "Chassis",
    "UUID": "48295861-2522-3561-6729-621118518810",
    ...
    "IndicatorLED": "Lit",
    ...
}
```

Redfish client can turn *IndicatorLED* into a different state:

```
curl -d '{"IndicatorLED": "Blinking"}' \
    -H "Content-Type: application/json" -X PATCH \
     http://localhost:8000/redfish/v1/Chassis/48295861-2522-3561-6729-
↪621118518810
```

Virtual media resource

The Virtual Media resource is emulated as a persistent emulator database record, observable and manageable by a Redfish client.

By default, a *VirtualMedia* resource includes two emulated removable devices: *Cd* and *Floppy*. Each *Manager* resource gets its own collection of virtual media devices as a *VirtualMedia* sub-resource.

If the currently used *Systems* resource emulation driver supports setting the boot image, the *VirtualMedia* resource will apply the inserted image onto all the systems being managed by this manager. Setting the system boot source to *Cd* and boot mode to *Uefi* will cause the system to boot from the virtual media image.

The user can change virtual media devices and their properties through emulator configuration (except for the OpenStack driver which only supports *Cd*):

```
SUSHY_EMULATOR_VMEDIA_DEVICES = {
    "Cd": {
        "Name": "Virtual CD",
        "MediaTypes": [
            "CD",
            "DVD"
        ]
    },
    "Floppy": {
        "Name": "Virtual Removable Media",
        "MediaTypes": [
            "Floppy",
            "USBStick"
        ]
    }
}
```

Virtual Media resource will be revealed when querying System resource:

```
curl -L http://localhost:8000/redfish/v1/Systems/58893887-8974-2487-2389-
↪841168418919/VirtualMedia
{
    "@odata.type": "#VirtualMediaCollection.VirtualMediaCollection",
    "Name": "Virtual Media Services",
    "Description": "Redfish-BMC Virtual Media Service Settings",
    "Members@odata.count": 2,
    "Members": [

        {
            "@odata.id": "/redfish/v1/Systems/58893887-8974-2487-2389-
↪841168418919/VirtualMedia/Cd"
        },

        {
            "@odata.id": "/redfish/v1/Systems/58893887-8974-2487-2389-
↪841168418919/VirtualMedia/Floppy"
        }
```

(continues on next page)

(continued from previous page)

```
    ],
    "@odata.context": "/redfish/v1/$metadata#VirtualMediaCollection.
↪VirtualMediaCollection",
    "@odata.id": "/redfish/v1/Systems/58893887-8974-2487-2389-841168418919/
↪VirtualMedia",
    "@Redfish.Copyright": "Copyright 2014-2017 Distributed Management Task␣
↪Force, Inc. (DMTF). For the full DMTF copyright policy, see http://www.
↪dmtf.org/about/policies/copyright."
}
```

Redfish client can insert a HTTP-based image into the virtual device:

```
curl -d '{"Image": "http://localhost.localdomain/mini.iso", "Inserted":␣
↪true}' \
     -H "Content-Type: application/json" \
     -X POST \
     http://localhost:8000/redfish/v1/Systems/58893887-8974-2487-2389-
↪841168418919/VirtualMedia/Cd/Actions/VirtualMedia.InsertMedia
```

On insert the OpenStack driver will:

- Upload the image directly to glance from the URL (long running)
- Store the URL, image ID and volume ID in server metadata properties *sushy-tools-image-url*, *sushy-tools-import-image*, *sushy-tools-volume*
- Create and attach a new volume with the same size as the root disk
- Rebuild the server with the image, replacing the contents of the root disk
- Delete the image

Redfish client can eject image from virtual media device:

```
curl -d '{}' \
     -H "Content-Type: application/json" \
     -X POST \
     http://localhost:8000/redfish/v1/Systems/58893887-8974-2487-2389-
↪841168418919/VirtualMedia/Cd/Actions/VirtualMedia.EjectMedia
```

On eject the OpenStack driver will:

- Assume the attached Volume has been rewritten with a new image (an ISO installer or IPA)
- Detach the Volume
- Create an image from the Volume (long running)
- Store the Volume image ID in server metadata property *sushy-tools-volume-image*
- Rebuild the server with the new image
- Delete the Volume
- Delete the image

Virtual media boot

To boot a system from a virtual media device, the client first needs to figure out which Manager is responsible for the system of interest:

```
$ curl http://localhost:8000/redfish/v1/Systems/281c2fc3-dd34-439a-9f0f-
↪63df45e2c998
{
...
"Links": {
    "Chassis": [
    ],
    "ManagedBy": [
        {
            "@odata.id": "/redfish/v1/Managers/58893887-8974-2487-2389-
↪841168418919"
        }
    ]
},
...
```

Exploring the Redfish API links, the client can learn the virtual media devices being offered:

```
$ curl http://localhost:8000/redfish/v1/Systems/58893887-894-2487-2389-
↪841168418919/VirtualMedia
...
"Members": [
{
    "@odata.id": "/redfish/v1/Systems/58893887-8974-2487-2389-841168418919/
↪VirtualMedia/Cd"
},
...
```

Knowing the virtual media device name, the client can check out its present state:

```
$ curl http://localhost:8000/redfish/v1/Systems/58893887-8974-2487-2389-
↪841168418919/VirtualMedia/Cd
{
    ...
    "Name": "Virtual CD",
    "MediaTypes": [
        "CD",
        "DVD"
    ],
    "Image": "",
    "ImageName": "",
    "ConnectedVia": "URI",
    "Inserted": false,
    "WriteProtected": false,
    ...
```

Assuming that the *http://localhost/var/tmp/mini.iso* URL points to a bootable UEFI or hybrid ISO, the following Redfish REST API call will insert the image into the virtual CD drive:

```
$ curl -d \
    '{"Image":"http:://localhost/var/tmp/mini.iso", "Inserted": true}' \
     -H "Content-Type: application/json" \
     -X POST \
     http://localhost:8000/redfish/v1/Systems/58893887-8974-2487-2389-
↪841168418919/VirtualMedia/Cd/Actions/VirtualMedia.InsertMedia
```

Querying again, the emulator should have it in the drive:

```
$ curl http://localhost:8000/redfish/v1/Systems/58893887-8974-2487-2389-
↪841168418919/VirtualMedia/Cd
{
    ...
    "Name": "Virtual CD",
    "MediaTypes": [
        "CD",
        "DVD"
    ],
    "Image": "http://localhost/var/tmp/mini.iso",
    "ImageName": "mini.iso",
    "ConnectedVia": "URI",
    "Inserted": true,
    "WriteProtected": true,
    ...
```

Next, the node needs to be configured to boot from its local CD drive over UEFI:

```
$ curl -X PATCH -H 'Content-Type: application/json' \
    -d '{
      "Boot": {
          "BootSourceOverrideTarget": "Cd",
          "BootSourceOverrideMode": "Uefi",
          "BootSourceOverrideEnabled": "Continuous"
      }
    }' \
    http://localhost:8000/redfish/v1/Systems/281c2fc3-dd34-439a-9f0f-
↪63df45e2c998
```

Note

With the OpenStack driver the boot source is changed during insert and eject, so setting *BootSourceOverrideTarget* to *Cd* or *Hdd* has no effect.

By this point the system will boot off the virtual CD drive when powering it on:

```
curl -d '{"ResetType":"On"}' \
    -H "Content-Type: application/json" -X POST \
     http://localhost:8000/redfish/v1/Systems/281c2fc3-dd34-439a-9f0f-
↪63df45e2c998/Actions/ComputerSystem.Reset
```

Note

The ISO files to boot from must be UEFI-bootable. libvirtd should be running on the same machine with sushy-emulator.

Storage resource

For emulating *Storage* resource for a System of choice, the user can statically configure one or more imaginary storage instances along with the corresponding storage controllers which are also imaginary.

The IDs of the imaginary drives associated with a *Storage* resource can be provided as a list under *Drives*.

The *Storage* instances are keyed by the UUIDs of the System they belong to.

```
SUSHY_EMULATOR_STORAGE = {
    "da69abcc-dae0-4913-9a7b-d344043097c0": [
        {
            "Id": "1",
            "Name": "Local Storage Controller",
            "StorageControllers": [
                {
                    "MemberId": "0",
                    "Name": "Contoso Integrated RAID",
                    "SpeedGbps": 12
                }
            ],
            "Drives": [
                "32ADF365C6C1B7BD"
            ]
        }
    ]
}
```

The Storage resources can be revealed by querying the Storage resource for the corresponding System directly.

```
curl http://localhost:8000/redfish/v1/Systems/da69abcc-dae0-4913-9a7b-
↪d344043097c0/Storage
{
    "@odata.type": "#StorageCollection.StorageCollection",
    "Name": "Storage Collection",
    "Members@odata.count": 1,
    "Members": [
        {
            "@odata.id": "/redfish/v1/Systems/da69abcc-dae0-4913-9a7b-
↪d344043097c0/Storage/1"
        }
    ],
    "Oem": {},
    "@odata.context": "/redfish/v1/$metadata#StorageCollection.
↪StorageCollection",
```

(continues on next page)

(continued from previous page)

```
    "@odata.id": "/redfish/v1/Systems/da69abcc-dae0-4913-9a7b-d344043097c0/
↪Storage"
}
```

Drive resource

For emulating the *Drive* resource, the user can statically configure one or more Drives.

The *Drive* instances are keyed in a composite manner using (System_UUID, Storage_ID), where System_UUID is the UUID of the System and Storage_ID is the ID of the Storage resource to which that particular Drive belongs.

```
SUSHY_EMULATOR_DRIVES = {
    ("da69abcc-dae0-4913-9a7b-d344043097c0", "1"): [
        {
            "Id": "32ADF365C6C1B7BD",
            "Name": "Drive Sample",
            "CapacityBytes": 899527000000,
            "Protocol": "SAS"
        }
    ]
}
```

The *Drive* resource can be revealed by querying it via the System and the Storage resource it belongs to.

```
curl http://localhost:8000/redfish/v1/Systems/da69abcc-dae0-4913-9a7b-
↪d344043097c0/Storage/1/Drives/32ADF365C6C1B7BD
{
    ...
    "Id": "32ADF365C6C1B7BD",
    "Name": "Drive Sample",
    "Model": "C123",
    "Revision": "100A",
    "CapacityBytes": 899527000000,
    "FailurePredicted": false,
    "Protocol": "SAS",
    "MediaType": "HDD",
    "Manufacturer": "Contoso",
    "SerialNumber": "1234570",
    ...
}
```

Storage Volume resource

The *Volume* resource is emulated as a persistent emulator database record, backed by the libvirt virtualization backend of the dynamic Redfish emulator.

Only the volumes specified in the config file or created via a POST request are allowed to be emulated upon by the emulator and appear as libvirt volumes in the libvirt virtualization backend. Volumes other than these can neither be listed nor deleted.

To allow libvirt volumes to be emulated upon, they need to be specified in the configuration file in the following format (keyed compositely by the System UUID and the Storage ID):

```
SUSHY_EMULATOR_VOLUMES = {
    ('da69abcc-dae0-4913-9a7b-d344043097c0', '1'): [
        {
            "libvirtPoolName": "sushyPool",
            "libvirtVolName": "testVol",
            "Id": "1",
            "Name": "Sample Volume 1",
            "VolumeType": "Mirrored",
            "CapacityBytes": 23748
        },
        {
            "libvirtPoolName": "sushyPool",
            "libvirtVolName": "testVol1",
            "Id": "2",
            "Name": "Sample Volume 2",
            "VolumeType": "StripedWithParity",
            "CapacityBytes": 48395
        }
    ]
}
```

The Volume resources can be revealed by querying the Volumes resource for the corresponding System and Storage.

```
curl http://localhost:8000/redfish/v1/Systems/da69abcc-dae0-4913-9a7b-
↪d344043097c0/Storage/1/Volumes
{
    "@odata.type": "#VolumeCollection.VolumeCollection",
    "Name": "Storage Volume Collection",
    "Members@odata.count": 2,
    "Members": [
        {
            "@odata.id": "/redfish/v1/Systems/da69abcc-dae0-4913-9a7b-
↪d344043097c0/Storage/1/Volumes/1"
        },
        {
            "@odata.id": "/redfish/v1/Systems/da69abcc-dae0-4913-9a7b-
↪d344043097c0/Storage/1/Volumes/2"
        }
    ],
    "@odata.context": "/redfish/v1/$metadata#VolumeCollection.
↪VolumeCollection",
    "@odata.id": "/redfish/v1/Systems/da69abcc-dae0-4913-9a7b-d344043097c0/
↪Storage/1/Volumes",
}
```

A new volume can also be created in the libvirt backend via a POST request on a Volume Collection:

```
curl -d '{"Name": "SampleVol",\
         "VolumeType": "Mirrored",\
         "CapacityBytes": 74859}' \
    -H "Content-Type: application/json" \
    -X POST \
    http://localhost:8000/redfish/v1/Systems/da69abcc-dae0-4913-9a7b-
↪d344043097c0/Storage/1/Volumes
```

1.4 Contributing

If you would like to contribute to the development of OpenStack, you must follow the steps in this page:

> http://docs.openstack.org/infra/manual/developers.html

If you already have a good understanding of how the system works and your OpenStack accounts are set up, you can skip to the development workflow section of this documentation to learn how changes to OpenStack should be submitted for review via the Gerrit tool:

> http://docs.openstack.org/infra/manual/developers.html#development-workflow

Pull requests submitted through GitHub will be ignored.

Bugs should be filed on Launchpad, not GitHub:

> https://bugs.launchpad.net/sushy

1.4.1 Cloning the sushy-tools repository

If you haven't already, the sushy-tools source code should be pulled directly from git.

```
# from the directory where you want the source code to reside
git clone https://opendev.org/openstack/sushy-tools
```

1.4.2 Running the emulators locally

Activate the virtual environment and run the emulator of your choice. For instance, to run the dynamic emulator:

```
tox -e venv -- sushy-emulator
```

For more information on running the emulators, refer to the user docs for the *dynamic-emulator* and the *static- emulator*.

CHAPTER TWO

INDICES AND TABLES

- genindex
- modindex
- search