sushy Documentation

Release

OpenStack Foundation

May 24, 2017

Contents

Contents:

CHAPTER 1

Sushy

Sushy is a Python library to communicate with Redfish based systems.

The goal of the library is to be extremely simple, small, have as few dependencies as possible and be very conservative when dealing with BMCs by issuing just enough requests to it (BMCs are very flaky).

Therefore, the scope of the library has been limited to what is supported by the OpenStack Ironic project. As the project grows and more features from Redfish are needed we can expand Sushy to fullfil those requirements.

- Free software: Apache license
- Documentation: http://sushy.rtfd.io
- Usage: http://sushy.readthedocs.io/en/latest/usage.html
- Source: http://git.openstack.org/cgit/openstack/sushy
- Bugs: http://bugs.launchpad.net/sushy

Features

- Abstraction around the SystemCollection and System resources (Basic server identification and asset information)
- Systems power management (Both soft and hard; Including NMI injection)
- Changing systems boot device, frequency (Once or permanently) and mode (UEFI or BIOS)

Check out the *Usage* page.

TODO

- Collect sensor data (Health state, temperature, fans etc...)
- System inspection (Number of CPUs, memory and disk size)

- Serial console

CHAPTER 2

Installation

At the command line:

```
$ pip install sushy
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv sushy
$ pip install sushy
```

CHAPTER 3

Usage

To use sushy in a project:

```
import logging

import sushy

# Enable logging at DEBUG level
LOG = logging.getLogger('sushy')
LOG.setLevel(logging.DEBUG)
LOG.addHandler(logging.StreamHandler())

s = sushy.Sushy('http://localhost:8000', username='foo', password='bar')

# Get the Redfish version
print(s.redfish_version)

# Instantiate a system object
sys_inst = s.get_system('/redfish/v1/Systems/437XR1138R2')


# Using system collections


# Instantiate a SystemCollection object
sys_col = s.get_system_collection()

# Print the ID of the systems available in the collection
print(sys_col.members_identities)

# Get a list of systems objects available in the collection
sys_col_insts = sys_col.get_members()

# Instantiate a system object, same as getting it directly
# from the s.get_system()
sys_inst = sys_col.get_member(sys_col.members_identities[0])
```

```
# Refresh the system collection object
sys_col.refresh()


# Using system actions


# Power the system ON
sys_inst.reset_system(sushy.RESET_ON)

# Get a list of allowed reset values
print(sys_inst.get_allowed_reset_system_values())

# Refresh the system object
sys_inst.refresh()

# Get the current power state
print(sys_inst.power_state)

# Set the next boot device to boot once from PXE in UEFI mode
sys_inst.set_system_boot_source(sushy.BOOT_SOURCE_TARGET_PXE,
                                enabled=sushy.BOOT_SOURCE_ENABLED_ONCE,
                                mode=sushy.BOOT_SOURCE_MODE_UEFI)

# Get the current boot source information
print(sys_inst.boot)

# Get a list of allowed boot source target values
print(sys_inst.get_allowed_system_boot_source_values())

# Get the memory summary
print(sys_inst.memory_summary)

# Get the processor summary
print(sys_inst.processors.summary)
```

If you do not have any real baremetal machine that supports the Redfish protocol you can look at the *Contributing* page to learn how to run a Redfish emulator.

CHAPTER 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

Running a Redfish emulator

Testing and/or developing Sushy without owning a real baremetal machine that supports the Redfish protocol is possible by running an emulator, the sushy-tools project ships with two emulators that can be used for this purpose. To install it run:

```
sudo pip install --user sushy-tools
```

Note: Installing the dependencies requires libvirt development files. For example, run the following command to install them on Fedora:

```
sudo dnf install -y libvirt-devel
```

Static emulator

After installing sushy-tools you will have a new CLI tool named `sushy-static`. This tool creates a HTTP server to serve any of the Redfish mockups. The files are static so operations like changing the boot device or the power state **will not** have any effect. But that should be enough for enabling people to test parts of the library.

To use `sushy-static` we need the Redfish mockup files that can be downloaded from https://www.dmtf.org/standards/redfish, for example:

```
wget https://www.dmtf.org/sites/default/files/standards/documents/DSP2043_1.0.0.zip
```

After the download, extract the files somewhere in the file-system:

```
unzip DSP2043_1.0.0.zip -d <output-path>
```

Now run `sushy-static` pointing to those files. For example to serve the `DSP2043-server` mockup files, run:

```
sushy-static --mockup-files <output-path>/DSP2043-server
```

Libvirt emulator

The second emulator shipped by sushy-tools is the CLI tool named `sushy-emulator`. This tool starts a ReST API that users can use to interact with virtual machines using the Redfish protocol. So operations such as changing the boot device or the power state will actually affect the virtual machines. This allows users to test the library in a more dynamic way. To run it do

```
sushy-emulator

# Or, running with custom parameters
sushy-emulator --port 8000 --libvirt-uri "qemu:///system"
```

That's it, now you can test Sushy against the `http://locahost:8000` endpoint.

Enabling SSL

Both mockup servers supports SSL if you want Sushy with it. To set it up, first you need to generate key and certificate files with OpenSSL use following command:

```
openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365
```

Start the mockup server passing the `--ssl-certificate` and `--ssl-key` parameters to it to it, for example:

```
sushy-emulator --ssl-key key.pem --ssl-certificate cert.pem
```

Now to connect with SSL to the server use the `verify` parameter pointing to the certificate file when instantiating Sushy, for example:

```
import sushy

# Note the HTTP"S"
s = sushy.Sushy('https://localhost:8000', verify='cert.pem', username='foo', password=
↪'bar')
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search