**NVIDIA Base Command Manager 10**

# Developer Manual

Revision: 1a3edea19

Date: Tue Jul 1 2025

## Trademarks

Linux is a registered trademark of Linus Torvalds. PathScale is a registered trademark of Cray, Inc. Red Hat and all Red Hat-based trademarks are trademarks or registered trademarks of Red Hat, Inc. SUSE is a registered trademark of SUSE LLC. NVIDIA, CUDA, GPUDirect, HPC SDK, NVIDIA DGX, NVIDIA Nsight, and NVLink are registered trademarks of NVIDIA Corporation. FLEXlm is a registered trademark of Flexera Software, Inc. PBS Professional, and Green Provisioning are trademarks of Altair Engineering, Inc. All other trademarks are the property of their respective owners.

## Rights and Restrictions

All statements, specifications, recommendations, and technical information contained herein are current or planned as of the date of publication of this document. They are reliable as of the time of this writing and are presented without warranty of any kind, expressed or implied. NVIDIA Corporation shall not be liable for technical or editorial errors or omissions which may occur in this document. NVIDIA Corporation shall not be liable for any damages resulting from the use of this document.

## Limitation of Liability and Damages Pertaining to NVIDIA Corporation

The NVIDIA Base Command Manager product principally consists of free software that is licensed by the Linux authors free of charge. NVIDIA Corporation shall have no liability nor will NVIDIA Corporation provide any warranty for the NVIDIA Base Command Manager to the extent that is permitted by law. Unless confirmed in writing, the Linux authors and/or third parties provide the program as is without any warranty, either expressed or implied, including, but not limited to, marketability or suitability for a specific purpose. The user of the NVIDIA Base Command Manager product shall accept the full risk for the quality or performance of the product. Should the product malfunction, the costs for repair, service, or correction will be borne by the user of the NVIDIA Base Command Manager product. No copyright owner or third party who has modified or distributed the program as permitted in this license shall be held liable for damages, including general or specific damages, damages caused by side effects or consequential damages, resulting from the use of the program or the un-usability of the program (including, but not limited to, loss of data, incorrect processing of data, losses that must be borne by you or others, or the inability of the program to work together with any other program), even if a copyright owner or third party had been advised about the possibility of such damages unless such copyright owner or third party has signed a writing to the contrary.

# Table of Contents

# Preface

Welcome to the *Developer Manual* for NVIDIA Base Command Manager 10.

## 0.1 About This Manual

This manual is aimed at helping developers who would like to program the NVIDIA Base Command Manager in order to enhance or alter its functionality. It is not intended for end users who simply wish to submit jobs that run programs to workload managers, which is discussed in the *User Manual*. The developer is expected to be reasonably familiar with the parts of the *Administrator Manual* that is to be dealt with—primarily CMDaemon, of which `cmsh` and Base View are the front ends.

This manual discusses the Python API to CMDaemon, and also covers how to program for metric collections.

## 0.2 About The Manuals In General

### Name Changes From Version 9.2 To 10

*The cluster manager software was originally developed by Bright Computing and the name "Bright" featured previously in the product, repositories, websites, and manuals.*

*Bright Computing was acquired by NVIDIA in 2022. The corresponding name changes, to be consistent with NVIDIA branding and products, are a work in progress. There is some catching up to do in places. For now, some parts of the manual still refer to Bright Computing and Bright Cluster Manager. These remnants will eventually disappear during updates.*

*BCM in particular is a convenient abbreviation that happens to have the same letters as the former Bright Cluster Manager. With the branding change in version 10, Base Command Manager is the official full name for the product formerly known as Bright Cluster Manager, and BCM is the official abbreviation for Base Command Manager.*

Regularly updated versions of the NVIDIA Base Command Manager 10 manuals are available on updated clusters by default at `/cm/shared/docs/cm`. The latest updates are always online at `https://docs.nvidia.com/base-command-manager`.

- The *Administrator Manual* describes the general management of the cluster.
- The *Installation Manual* describes installation procedures for a basic cluster.
- The *User Manual* describes the user environment and how to submit jobs for the end user.
- The *Cloudbursting Manual* describes how to deploy the cloud capabilities of the cluster.
- The *Developer Manual* has useful information for developers who would like to program with BCM.
- The *Edge Manual* describes how to deploy BCM Edge with BCM.
- The *Machine Learning Manual* describes how to install and configure machine learning capabilities with BCM.
- The *Containerization Manual* describes how to manage containers with BCM.

If the manuals are downloaded and kept in one local directory, then in most pdf viewers, clicking on a cross-reference in one manual that refers to a section in another manual opens and displays that section in the second manual. Navigating back and forth between documents is usually possible with keystrokes or mouse clicks.

For example: <Alt>-<Backarrow> in Acrobat Reader, or clicking on the bottom leftmost navigation button of xpdf, both navigate back to the previous document.

The manuals constantly evolve to keep up with the development of the BCM environment and the addition of new hardware and/or applications. The manuals also regularly incorporate feedback from administrators and users, and any comments, suggestions or corrections will be very gratefully accepted at manuals@brightcomputing.com.

There is also a feedback form available via Base View, via the menu icon, ☰, following the navigation path:

☰>Help>Feedback

## 0.3 Getting Administrator-Level Support

Support for BCM subscriptions from version 10 onwards is available via the NVIDIA Enterprise Support page at:

https://www.nvidia.com/en-us/support/enterprise/

Section 16.2 of the *Administrator Manual* has more details on working with support.

## 0.4 Getting Developer-Level Support

Developer support is given free, within reason. For more extensive support, the BCM support team can be contacted in order to arrange a support contract.

## 0.5 Getting Professional Services

The BCM support team normally differentiates between

- regular support (customer has a question or problem that requires an answer or resolution), and
- professional services (customer asks for the team to do something or asks the team to provide some service).

Professional services can be provided via the NVIDIA Enterprise Services page at:

https://www.nvidia.com/en-us/support/enterprise/services/

# 1

# NVIDIA Base Command Manager Python API

This chapter introduces the Python API of NVIDIA Base Command Manager.

The Python API package was completely overhauled in NVIDIA Base Command Manager 8.2.

The `cmdaemon-pythoncm` package now provides a pure Python connection to the cluster manager, making it possible for cluster administrators to automate cluster operations via Python.

It also makes it possible to run Python code on any operating system that supports Python 3.5 and higher.

The BCM Python API uses the following extra modules:

1. `pyOpenSSL`
2. `ply`
3. `lxml`
4. `tabulate`
5. `monotonic`
6. `humanfriendly`
7. `pyYAML`
8. `six`

## 1.1 Getting Started

On the cluster head node itself the `python3` module can simply be loaded:

**Example**

```
[root@basecm10 ~]# module load python3
```

To execute or develop the Python code on any other machine usually requires some extra steps:

- Python 3 (3.5, 3.6, 3.7, 3.8 will work) should be installed
- The 8 extra modules listed previously should be installed using Pip
- The `/cm/local/apps/cmd/pythoncm/lib/python3.9/site-packages/pythoncm` directory should be copied over to the site-packages directory of the development machine. The version is enforced in `lib/python3.9/site-packages/pythoncm/__init__.py`, so version consistency would be required here.

The `pythoncm` module should then be loaded, to confirm everything was set up correctly:

**Example**

```
[alice@desktop ~]# python -c "import pythoncm"
```

If connecting from outside the cluster, then port 8081 must not be blocked by a firewall.

A certificate is needed by the Python API to identify itself to `CMDaemon`.

The existence of the certificate on the head node should be checked. It should be copied over to the development machine, if it is needed there.

**Example**

```
[root@basecm10 ~]# ls -al .cm/
-rw------- 1 root root 1708 Dec 11 09:25 admin.key
-rw------- 1 root root 1269 Dec 11 09:25 admin.pem
```

**Example**

```
[alice@basecm10 ~]$ ls -al .cm/
-rw------- 1 root root 1708 Dec 11 09:25 cert.key
-rw------- 1 root root 1269 Dec 11 09:25 cert.pem
```

The developer many need to contact the cluster administrator to get a certificate.

## 1.2 Connecting To A Cluster

The first step when working with the Python API is to establish a connection to the `CMDaemon` process on the cluster:

```
#!/usr/bin/env python

from pythoncm.cluster import Cluster
from pythoncm.settings import Settings

cluster = Cluster()
```

If working outside the cluster, then the settings for connecting to the cluster must be specified:

```
settings = Settings(host='<head-node-hostname>',
                    port=8081,
                    cert_file='/some/path/cert.pem',
                    key_file='/some/path/cert.key',
                    ca_file='.../site-packages/pythoncm/etc/cacert.pem')
if not settings.check_certificate_files():
    print('Unable to load certificates')
else:
    cluster = Cluster(settings)
```

## 1.3 Inspecting Settings

All settings in BCM are stored inside an entity.

Each entity has a type and a unique name among the entities of the same type.

To inspect an entity it should first be found inside the cluster:

```
node001 = cluster.get_by_name('node001')
```

If the name `node001` was also given a different entity, then the type must be specified to ensure that the correct entity is returned:

```
node001 = cluster.get_by_name('node001', pythoncm.entity.Node)
node001 = cluster.get_by_name('node001', 'Node')
```

Once the node entity is found, inspecting the settings is a matter of printing the desired field:

```
print(node001.hostname)
print(node001.mac)
```

Complex settings, such as network interfaces, have their own settings:

```
for interfaces in node001.interfaces:
    print(interface.name, interface.ip)
```

Because many nodes could have a network interface called `eth0`, such a setting cannot be found from the cluster: The following code will return `None`.

```
eth0 = cluster.get_by_name('eth0')
```

To find all `eth0` interfaces, all nodes need to be found, and then iterated over:

```
nodes = cluster.get_by_type(pythoncm.entity.Node)
all_eth0 = [interface
            for node in nodes
            for interface in node.interfaces
            if interface.name == 'eth0']
```

## 1.4 Modifying Settings

Basic entity settings are exported as Python properties and can simply be changed:

```
node001.mac = '00:00:00:00:00:00'
node001.category = cluster.get_by_name('gpu', 'Category')
```

Similarly interfaces settings can be accessed and changed directly:

```
node001.interfaces[0].ip = '1.2.3.4'
node001.interfaces[0].network = cluster.get_by_name('ib', 'Network')
```

Removing an interface from a node can be done in various Pythonic ways:

```
node001.interfaces.remove(0)
del node001.interfaces[0]
node001.interfaces = [interface for interface in node001.interfaces
                      if interface.name != 'eth0']
```

To add a new interface, the entity instance needs to be created first, and then added to the node:

```
eth1 = pythoncm.entity.NetworkPhysicalInterface()
eth1.name = 'eth1'
eth1.ip = '1.2.3.4'
eth1.network = cluster.get_by_name('ib', 'Network')
node001.interfaces.append(eth1)
```

All changes are made on a local copy of the entity. The cluster has no knowledge of the changes until they are committed.

It is recommended to make many changes locally, and only commit once at the end.

The return value of the commit operation should always be checked.

Committing a badly-configured node will be blocked by the head node:

```
commit_result = node001.commit()
if not commit_result.good:
    print(commit_result)
```

An entity found from via the `cluster` object is removed differently.

As with commit, the result should always be checked: a removal can fail if a node is UP:

```
remove_result = node001.remove()
if not remove_result.good:
    print(remove_result)
```

## 1.5 Inspecting The Entire Cluster

The example directory contains a script to inspect the entire cluster.

**Example**

```
[root@basecm10 ~]# cd /cm/local/examples/cmd/pythoncm/
[root@basecm10 pythoncm]# module load python3
[root@basecm10 pythoncm]# ./print-all.py
```

The example directory also contains a script that prints all metadata for all available entities in BCM:

**Example**

```
[root@basecm10 pythoncm]# ./entity_info.py
```

## 1.6 Performing Operations On Entities

All Python API functionality is contained in `/cm/local/apps/cmd/pythoncm/lib/python3.9/site-packages/pythoncm`.

Methods are documented inside the python code itself.

```
node001 = cluster.get_by_name('node001')
node001.power_on()
```

When operating on multiple entities, it is possible to iterate over them and do each operation individually.

```
nodes = cluster.get_by_type('Node')
for node in nodes:
    node.power_on()
```

However the same can also be done with a parallel version of the operation. When possible the parallel version should be used, because it is faster and requires less network traffic.

```
nodes = cluster.get_by_type('Node')
cluster.parallel.power_on(nodes)
```

## 1.7 Monitoring

All monitoring data can be accessed using the Python API.

Monitoring is a set of operations performed on entities.

For example, to get latest data for a single entity:

```
print(node001.get_latest_monitoring_data())
```

Monitoring operations on multiple operations should be done using the `monitoring` module:

```
data = cluster.monitoring.get_latest_monitoring_data([node001,
                                                      node002,
                                                      node003])
```

## 1.8 Examples

The best way to get going is by looking at the examples. These can be found on the head node, at `/cm/local/examples/cmd/pythoncm`:

### Example

```
[root@basecm10 pythoncm]# ls
add-collection.py                    dump-job-monitoring-data.py        power-history.py
add-healthcheck.py                   dump-monitoring-data.py            power-parallel-status.py
add-metric.py                        entity_info.py                     power-status.py
add-node-group.py                    execute.py                         print-all.py
add-role.py                          fabric-bindings.py                 range-expander-test.py
add-user.py                          fake-file-write.py                 remove-many-nodes.py
all-background-tasks.py              free_port.py                       remove-node-group.py
all-nodes.py                         get-all-wlm-jobs.py                sample-all-checks.py
arch_os_image_info.py                get-status.py                      sample-now-checks.py
certificate-info.py                  health-overview.py                 sample-now-parallel.py
charge-back.py                       instance_by_name.py                select-devices.py
clone-many-nodes.py                  instant-query.py                   service.py
clone-node-group.py                  key_value_pair.py                  service-status.py
clone-node.py                        latest-counter-data.py             set-node-image.py
cm-job-analytics.py                  latest-health-data.py              test-add-update-remove.py
cm-job-gpu.py                        monitoring-push.py                 tftpboot-file-information.py
cm-network-traffic-monitor-setup.py  move-to-new-pdu.py                 total-job-power-usage.py
config-writer.py                     new-nodes.py                       up-percentage.py
cookie.py                            parallel-execute-async.py          user-data.py
create-certificate.py                parallel-execute-check-status.py   wait-for-provisioning.py
create-ramdisk-task.py               parallel-execute.py                wait-for-up.py
```

The examples can be tried out after loading the Python environment:

### Example

```
[root@basecm10 ~]# cd /cm/local/examples/cmd/pythoncm/
[root@basecm10 pythoncm]# module load python3
[root@basecm10 pythoncm]# ./power-status.py
INFO     (25-May-2020 18:29:25) [cluster.py          :207] Follow redirection to active head IP:
10.141.255.254
INFO     (25-May-2020 18:29:25) [cluster.py          :298] Start event thread for session
42949672967
success: True
[
    "uniqueKey": 1125899906842642,
    "oldLocalUniqueKey": 0,
    "baseType": "PowerStatus",
    "childType": "",
    "revision": "",
    "modified": false,
    "toBeRemoved": false,
    "readonly": false,
    "not_set_fields": [],
    "device": 38654705666,
    "host": 38654705665,
    "powerDistributionUnit": 0,
    "gpu": -1,
    "prt": 0,
```

```
    "name": "custom",
    "state": "ON",
    "msg": "",
    "extendedMsg": "",
    "indexes": [
        2
    ],
    "tracker": 0,
    "retries": 0
]
INFO     (25-May-2020 18:29:28) [entity_change.py    : 38] Stop event change watcher
[root@basecm10 pythoncm]#
```

# 2

# Monitoring Data Producers

This chapter covers how to add a new metrics and health checks scripts with `cmsh`.

Five different types of Monitoring Data Producers can be added:

- `metric`: a script which produces a single value.
- `health check`: a script which produces a `PASS`, `FAIL`, `UNKNOWN`, or `no data` value.
- `collection`: a script that produces zero or more metrics, health checks, or a combination of both.
- `perpetual` a script that is started once over the lifetime of the BCM `cmd` process. The script produces zero or more metrics, health checks, or a combination of both on its own timing mechanism.
- `prometheus` one or more URLs to Prometheus metric exporters.

A monitoring data producer cannot be plotted in `cmsh` or Base View, because it contains no data. A producer defines measurables: metrics and/or health checks. It also generates data for these measurables, which can be plotted.

## 2.1 Measurables

There are three types of measurable:

- `metric`: a numeric value, or `no data`.
- `health check`: `PASS/FAIL/UNKNOWN/no data`.
- `enum metric`: one of a set of user-defined string based values, or `no data`.

## 2.2 Measurables Classes

All measurables are grouped into classes. A class is a user-defined free string field, with / as delimiters. Base View uses this class to build a tree for easy search and access.

## 2.3 Metric Monitoring Data Producers

A metric data producer script generates one data point.

For example, as in the following script:

**Example**

```
[root@basecm10 ~]# cat /path/to/my/metric
#!/bin/bash
info_fd=${CMD_INFO_FD:=3}    #set CMD_INFO_FD to 3 by default only if nothing set
```

```
# CMD_INFO_FD:=3    <--- same as: if [ -z "$CMD_INFO_FD" ]; then CMD_INFO_FD=3; fi
echo $((RANDOM))
# Optionally provide extra information
echo "Extra information" >&$info_fd
```

The script can be defined as a metric script via the `monitoring setup` mode of `cmsh`:

**Example**

```
[basecm10]% monitoring setup
[basecm10->monitoring->setup]% add metric my-metric
[...my-metric]% set script /path/to/my/metric
[...my-metric]% set class My/Class
[...my-metric]% set unit B
[...my-metric]% set interval 1m
[...my-metric]% commit
```

All nodes then execute the script every minute, and produce a random number.

## 2.4 Health Check Monitoring Data Producers

A health check data producer script generates one data point. The data point can be one of four possible values expected of it: `PASS`, `FAIL`, `UNKNOWN`, or `no data`. Other file descriptors can be used to provide extra information.

For example, as in the following script:

**Example**

```
[root@basecm10 ~]# cat /path/to/my/health-check
#!/bin/bash
info_fd=${CMD_INFO_FD:=3}    #set CMD_INFO_FD to 3 by default if nothing set
if [ $((RANDOM)) -gt 8000 ]; then
  echo "PASS"
else
  echo "FAIL"
  # Optionally provide extra information
echo "Extra information" >&$info_fd
fi
```

The script can be defined as a health check script via the `monitoring setup` mode of `cmsh`:

**Example**

```
[basecm10]% monitoring setup
[basecm10->monitoring->setup]% add healthcheck my-health-check
[...my-check]% set script /path/to/my/health-check
[...my-check]% set class My/Class
[...my-check]% set interval 1m
[...my-check]% commit
```

All nodes then execute the script every minute, and produce data values with roughly 75% `PASS` and 25% `FAIL`.

## 2.5 Collection Monitoring Data Producers

A *collection data producer* script can generate multiple data points in one run. Data points can be a combination of metrics and health checks. Collection scripts are also allowed to produce no data.

A collection script has two modes: initialize mode and sample mode.

- `initialize`: defines the measurables that data values are generated for.
- `sample`: returns the data values for all the measurables defined in initialize mode.

During normal cluster operation the initialize mode is called only once, during boot. Afterwards, the script is called in sample mode at the desired interval.

The following example combines both of the metric and health check examples from earlier on. However, this time it is written as a single script, using JSON as the output format:

**Example**

```
[root@basecm10 ~]# cat /path/to/my/collection
#!/usr/bin/python

import sys
import json
import random

def initialize():
    metric = {"metric": "my.collection.metric",
              "unit": "B",
              "class": "My/Collection"}
    check = {"check": "my.collection.check",
             "class": "My/Collection"}
    return [metric, check]

def sample():
    metric = {"metric": "my.collection.metric",
              "value": random.randint(0, 32767)}
    check = {"check": "my.collection.check",
             "info": "random with 25% failure rate",
             "value": 'PASS' if random.randint(0, 32767) > 8000 else 'FAIL'}
    return [metric, check]

def main():
    if len(sys.argv) > 1 and sys.argv[1] == "--initialize":
        data = initialize()
    else:
        data = sample()
    print (json.dumps(data, indent=4))

if __name__ == '__main__':
    main()
```

The script can be defined as a collection script via the `monitoring setup` mode of `cmsh`:

**Example**

```
[basecm10]% monitoring setup
[basecm10->monitoring->setup]% add collection my-collection
[...my-collection]% set script /path/to/my/collection
[...my-collection]% set format JSON
```

```
[...my-collection]% set interval 1m
[...my-collection]% commit
```

All nodes then execute the script every minute and produce two data points upon each execution. That is, one metric and one health check per execution.

## 2.6 Perpetual Monitoring Data Producers

A perpetual data producer script is a special case of a collection data producer script. It is intended to be used if the script needs permanent memory storage.

### Example

```
[root@basecm10 ~]# cat /path/to/my/perpetual
#!/usr/bin/python

import my_sampler_module
import json
import time

# create single instance
sampler = my_sampler_module.MySampler()
# load important data into memory
sampler.load()

# Infinite loop with its own timing
delay = 0
while True:
    time.sleep(delay)
    (definitions, values, delay) = sampler.process()
    if definitions:
        # Print new measurables
        print (json.dumps(definitions))
    # Print data
    print (json.dumps(values))
```

The `my_sampler_module` is the part which does the important work.

### Example

```
[root@basecm10 ~]# cat /path/to/my/my_sampler_module.py
class MySampler:
    def __init__(self):
        self.initialized = False
        self.definitions = None

    def load(self):
        # Do time consuming work here
        metric = {"metric": "my.collection.metric",
                  "unit": "B",
                  "class": "My/Collection"}
        check = {"check": "my.collection.check",
                 "class": "My/Collection"}
        self.definitions = [metric, check]

    def process(self):
        metric = {"metric": "my.collection.metric",
```

```
                 "value": random.randint(0, 32767)}
        check = {"check" : "my.collection.check",
                 "value" : 'PASS' if random.randint(0, 32767) > 8000 else 'FAIL'}
        values = metric, check
        # return definitions once, afterwards they never change
        # but new definitions could be added this way
        definitions = self.definitions
        self.definitions = None
        return definitions, values, 60
```

The script can be defined as a perpetual script via the `monitoring setup` mode of `cmsh`:

**Example**

```
[basecm10]% monitoring setup
[basecm10->monitoring->setup]% add perpetual my-perpetual
[...my-perpetual]% set script /path/to/my/perpetual
[...my-perpetual]% set format JSON
[...my-perpetual]% commit
```

## 2.7 Prometheus Monitoring Data Producers

Prometheus is a monitoring and alerting toolkit (`https://prometheus.io`). A Prometheus monitoring data producer script parses data from a Prometheus exporter (`https://prometheus.io/docs/instrumenting/exporters/`)

The script can be defined as a Prometheus script via the `monitoring setup` mode of `cmsh`:

**Example**

```
[basecm10]% monitoring setup
[basecm10->monitoring->setup]% add prometheus my-prometheus-exporter
[...my-prometheus-exporter]% set urls http://my.prometheus.exporter:80
[...my-prometheus-exporter]% set interval 1m
[...my-prometheus-exporter]% commit
```

If multiple URLs are defined, then only the data values from the first successful `HTTP GET` are used.

## 2.8 Node Execution Filters

By default a monitoring data producer script is executed on every node. When this is not desirable, a node execution filter should be created. A node execution filter defines the nodes on which the producer script should be executed.

For example, a filter to execute the script only on cloud nodes can be configured as follows:

**Example**

```
[basecm10]% monitoring setup use my-check
[...my-check]% nodeexecutionfilters
[...nodeexecutionfilters]% add type Cloud
[...nodeexecutionfilters*[Cloud*]]% set cloudnode yes
[...nodeexecutionfilters*[Cloud*]]% show
Parameter          Value
------------------ ----------------------------
Base type          MonitoringExecutionFilter
Name               Cloud
Type               Type
Head node          no
```

```
Physical node      no
Cloud node         yes
Virtual node       no
Lite node          no
[...nodeexecutionfilters*[Cloud*]]% commit
```

It is also possible to filter based on the specific resources associated with a node:

**Example**

```
[basecm10]% monitoring setup use my-IB-check
[...my-IB-check]% nodeexecutionfilters
[...nodeexecutionfilters]% add resource IB
[...nodeexecutionfilters*[IB*]]% set resources IB
[...nodeexecutionfilters*[IB*]]% commit
```

Because of high availability, a special resource, `active`, is defined for the active head node.

**Example**

```
[basecm10]% monitoring setup use my-metric
[...my-metric]% nodeexecutionfilters
[...nodeexecutionfilters]% active
Added active resource filter
[...nodeexecutionfilters*]% commit
```

## 2.9 Execution Multiplexers

By default a monitoring data producer script is executed once: the node executes the script only for itself.

However, some scripts, such as BMC samplers, must be sampled from the active head node for all nodes.

In the following example a BMC script is run on each node that has the `ipmi` or `drac` resource:

**Example**

```
[basecm10]% monitoring setup use my-ipmi-collection
[...my-ipmi-collection]% executionmultiplexers
[...executionmultiplexers]% add resource ipmi
[...executionmultiplexers*[ipmi*]]% set resources ipmi drac
[...executionmultiplexers*[ipmi*]]% set operator OR
[...executionmultiplexers*[ipmi*]]% commit
```

If an execution multiplexer <*multiplexer*> is defined, then there should also be a node execution filter <*filter*> associated with it to restrict the number of nodes on which the script runs.

This is because having the script run on many nodes for many other nodes is unlikely to be a desired configuration.

The combination of the execution filter and the multiplexer should be read as:

for every node that matches *filter*, run script, for each node that matches *multiplexer*.

A more specific example, using two of the preceding examples, with a filter based on the resource `IB`, and multiplexers based on the IPMI/Drac resources, the combination should be read as:

for every node that matches `IB`, run script, for each node that matches `ipmi` or `drac`.

## 2.10 Monitoring Resources

Every device in BCM has one or more resources. These resources are automatically calculated from:

- Roles
- Hardware
- Settings

Resources for a specific node can be viewed as follows:

**Example**

```
[basecm10]% device use node001
[basecm10]% monitoringresources
CentOS7u5
Ethernet
category:default
```

It is possible to add one or more custom resources to a device:

**Example**

```
[basecm10]% device use node001
[basecm10]% add userdefinedresources MyResource
[basecm10]% append userdefinedresources MyOtherResource
[basecm10]% # wait ~10 seconds for the settings to propagate
[basecm10]% monitoringresources
CentOS7u5
Ethernet
category:default
MyResource
MyOtherResource
```

Any of these resources can be used to filter and multiplex monitoring data producers.

If a resources changes because of a settings change, then monitoring automatically stops or starts sampling.

## 2.11 Collection Monitoring Data Producers With Filter And Multiplexer

If a script has an execution multiplexer set, then it needs to determine for which nodes the script runs:

**Example**

```
[root@basecm10~]# cat /path/to/my/collection
#!/usr/bin/python

import sys
import json
import random

def initialize(entity):
    metric = {"metric": "my.collection.metric",
              "entity": entity,
              "unit": "B",
              "class": "My/Collection"}
    check = {"check": "my.collection.check",
             "entity": entity,
```

```
            "class": "My/Collection"}
    return [metric, check]

def sample(entity):
    metric = {"metric": "my.collection.metric",
              "entity": entity,
              "value": random.randint(0, 32767)}
    check = {"check" : "my.collection.check",
             "entity": entity,
             "value" : 'PASS' if random.randint(0, 32767) > 8000 else 'FAIL'}
    return [metric, check]

def main():
    try:
        # determine for which node we are sampling
        entity = os.environ['CMD_HOSTNAME']
    except:
        sys.stderr.write('Target device not specified in environment\n')
        return

    if len(sys.argv) > 1 and sys.argv[1] == "--initialize":
        data = initialize(entity)
    else:
        data = sample(entity)
    print (json.dumps(data, indent=4))

if __name__ == '__main__':
    main()
```

It can be defined with a filter to run on the active head for all nodes in the GPU category:

**Example**

```
[basecm10]% monitoring setup
[basecm10->monitoring->setup]% add collection my-collection
[...my-collection]% set script /path/to/my/collection
[...my-collection]% set format JSON
[...my-collection]% set interval 1m
[...my-collection]% nodeexecutionfilters
[...nodeexecutionfilters]% active
Added active resource filter
[...nodeexecutionfilters]% exit
[...my-collection]% executionmultiplexers
[...executionmultiplexers]% add category
[...executionmultiplexers*[GPU*]% add category GPU
[...executionmultiplexers*[GPU*]% commit
```

The script is then executed on the head, once for each node in the category of `GPU`.

## 2.12 Collection Monitoring Data Producers For Standalone Entities

Sometimes monitoring data does not belong to a BCM entity.

For this reason the standalone monitored entity was added in NVIDIA Base Command Manager 8.0.

This entity can be anything with a name and custom type.

BCM does nothing with this kind of entity, except allow it to store monitoring data.

Each standalone entity which needs to be monitored should be added:

### Example

```
[basecm10]% monitoring standalone
[basecm10->monitoring->standalone]% add MSD.0
[...standalone*[MSD.0*]]% set type Lustre
[...standalone*[MSD.0*]]% commit
[...standalone*[MSD.0*]]% add MSD.1
[...standalone*[MSD.1*]]% set type Lustre
[...standalone*[MSD.1*]]% commit
```

A script can be created that produces data for all MSD entities:

### Example

```
[root@basecm10 ~]# cat /path/to/my/collection
#!/usr/bin/python

import sys
import json

def initialize():
    msd_0 = {"metric": "lustre.free.space",
             "entity": "MSD.0",
             "unit": "B",
             "class": "Lustre"}
    msd_1 = {"metric": "lustre.free.space",
             "entity": "MSD.1",
             "unit": "B",
             "class": "Lustre"}
    return [msd_0, msd_1]

def sample():
    msd_0 = {"metric": "lustre.free.space",
             "entity": "MSD.0",
             "value": 12345,
             "class": "Lustre"}
    msd_1 = {"metric": "lustre.free.space",
             "entity": "MSD.1",
             "value": 54321}
    return [msd_0, msd_1]

def main():
    if len(sys.argv) > 1 and sys.argv[1] == "--initialize":
        data = initialize()
    else:
        data = sample()
    print (json.dumps(data, indent=4))

if __name__ == '__main__':
    main()
```

It can be defined to run on only the active head node:

### Example

```
[basecm10]% monitoring setup
[basecm10->monitoring->setup]% add collection my-collection
```

```
[...my-collection]% set script /path/to/my/collection
[...my-collection]% set format JSON
[...my-collection]% set interval 5m
[...my-collection]% nodeexecutionfilters
[...nodeexecutionfilters]% active
Added active resource filter
[...nodeexecutionfilters]% commit
```

The script is then executed on the active head every 5 minutes and collects one data point for each MSD.

Data for a standalone script can be viewed with the same commands as for regular BCM nodes.

**Example**

```
[basecm10]% monitoring standalone
[basecm10->monitoring->standalone]% use MSD.0
[...standalone*[MSD.0*]]% latestmetricdata
...
lustre.free.space        12345          3m 47s
```

## 2.13 Debugging Standalone Scripts

Page 637 of the *Administrator Manual* describes how debugging information can be obtained when running `samplenow` with the `--debug` option with the `ntp` healthcheck script.

Many scripts under `/cm/local/apps/cmd/scripts/` can have their debug output inspected with `samplenow --debug`.

A recursive grep on the head node, similar to the following, should show which scripts have a settable debug environment:

```
grep -r CMD_DEBUG /cm/local/apps/cmd/scripts/
```

The debug output in the script can be specified along the lines of the following code snippet:

**Example**

```
import os
debug = os.environ.get('CMD_DEBUG', '0') == '1'
info_fd = int(os.environ.get('CMD_INFO_FD', '3'))
if debug:
    os.write(info_fd, 'debug message')
```

# 3

# Monitoring Actions

This chapter covers how to manage monitoring-driven actions with `cmsh`.

## 3.1 Actions And Triggers

A monitoring action is a script that is executed by CMDaemon. It runs when triggered by the monitored data.

An action by itself does nothing—it needs a trigger (section 12.4.5 of the *Administrator Manual*) to be defined to execute the action.

By default, several actions (section 12.4.4 of the *Administrator Manual*) are predefined:

- `Drain`: Drain node (node refuses new WLM jobs)
- `Event`: Send an event to users with connected client
- `ImageUpdate`: Update the image on the node
- `PowerOff`: Power off a device
- `PowerOn`: Power on a device:
- `PowerReset`: Power reset a device
- `Reboot`: Reboot a node
- `Send e-mail to administrators`: Send e-mail
- `Shutdown`: Shutdown a node
- `Undrain`: Undrain node (node accepts new WLM jobs)
- `killprocess`: `/cm/local/apps/cmd/scripts/actions/killprocess.pl`
- `remount`: `/cm/local/apps/cmd/scripts/actions/remount`
- `testaction`: `/cm/local/apps/cmd/scripts/actions/testaction`

A new action script can be created as follows:

**Example**

```
[basecm10]% monitoring action
[basecm10->monitoring->action]% add script MyScript
[...MyScript*]% set script /path/to/MyScript
[...MyScript*]% commit
```

## 3.2 Time Restrictions

It is possible to allow actions to only be executed at certain times, with the `allowedtime` setting.

#### Example

```
[basecm10]% monitoring action
[basecm10->monitoring->action]% add script MyScript
[...MyScript*]% set script /path/to/MyScript
[...MyScript*]% set allowedtime "9:00-17:00"
[...MyScript*]% commit
```

More complex timing restrictions are possible:

#### Example

```
monday-friday{9:00-17:00}
monday-friday{00:00-09:00;17:00-00:00};saturday-sunday
november-march{monday-saturday{13:00-17:00}}
may-september{monday-friday{09:00-18:00};saturday-sunday{13:00-17:00}}
```

Further examples can be seen in section 12.4.4 of the *Administrator Manual*, page 608.

### 3.2.1 Time Restriction Syntax In BNF Notation

The allowed values can be written as a BNF grammar:

#### Example

```
<start> =
  time_intervals
  | ""
<time_intervals> = <time_interval> (; <time_interval>)*
<time_interval> = <inner_time_interval>{<time_intervals}
<inner_time_interval> =
  <day_of_week_interval>
  | <time_of_day_interval>
  | <day_of_month_interval>
  | <month_interval>
<day_of_week_interval> =
  (<day_of_week>-<day_of_week>)
  | (<day_of_week> (, day_of_week)*)
<day_of_week> = sunday | monday | tuesday | wednesday | thursday | friday | saturday
<time_of_day_interval> = <time_of_day>-<time_of_day>
<time_of_day>= \d?\d:\d\d
<month_interval> = (<month>-<month>)
  | (<month> (, month)*)
<month> = january | february | march | april | may | june | july | august | september
  | october | november | december
<day_of_month_interval> = (<day_of_month>-<day_of_month>)
  | (<day_of_month> (, day_of_month)*)
<day_of_month> = \d?\d
```

## 3.3 CMDaemon Environment Variables

### 3.3.1 Standard Environment Variables Available In Action Scripts

| Name | Description |
|---|---|
| CMD_ENTITY_UUID | The UUID of the entity that triggered the action. |
| CMD_ENTITY_NAME | The name of the entity that triggered the action. |
| CMD_ENTITY_TYPE | The type of entity that triggered the action. |
| CMD_MEASURABLE_NAME | The name of the measurable that triggered the action. |
| CMD_MEASURABLE_PARAMETER | The parameter of the measurable that triggered the action. |
| CMD_MEASURABLE_TYPE | The type of the measurable. |
| CMD_VALUE | The value that triggered the action. |
| CMD_RAW_VALUE | The raw value. |
| CMD_VALUE_TIME | The time on which the value was measured, in ms elapsed since Unix epoch. |
| CMD_INFO_MESSAGE | Extra information sampled along with the value. |
| CMD_PRODUCER_NAME | The name of the monitoring data producer that samples the measurable. |

*...continues*

*...continued*

| Name | Description |
|---|---|
| CMD_ACTION_NAME | The name of the action that was triggered. |
| CMD_TRIGGER_NAME | The name of the trigger. |
| CMD_TRIGGER_EXPRESSION | The expression that was evaluated. |
| CMD_VALUE_EVAL | The result of the evaluated expression. |
| CMD_VALUE_COUNT | The number of times the expression evaluated to the same value. |
| CMD_SEVERITY | The assigned severity of the trigger. |
| CMD_STATE_FLAPPING | Is the measurable flapping? |
| CMD_JSON_DATA | If set to `yes`, then a JSON blob is passed on STDIN. The blob contains the information normally in `env`. `CMD_JSON_DATA` is used by the `Send e-mail` action |
| CMD_MEASURABLE_UUID | The UUID of the measurable used |
| CMD_MULTI_ACTION | Data values from measurables or triggers are dealt with together in some way if the action is `yes` |

### Example

If the value of the parameter `mergedmeasurables` is set to `yes`, and if more than one data value is gathered within a period that is specified by the value of `mergedelay`, then the data values for a measurable are dealt with together in some way by the action.
`Merge delay`, `Merge triggers`, and `Merge measurables` are parameters that are available within some monitoring actions.

```
[basecm10->monitoring->action]% list -f name:0,mergedelay,mergemeasurables
name (key) mergedelay mergemeasurables
--------------- ---------- ----------
...
Send e-mail to administrators 2s yes
...
```

The `Send e-mail` action in this supports `CMD_MULTI_ACTION=yes` by sending one email with multiple values, instead of one e-mail per measured or triggered value:

*...continues*

*...continued*

| Name | Description |
|---|---|
| CMD_TRIGGER_UUID | UUID of the trigger |
| CMD_SCRIPTTIMEOUT | script timeout, in seconds, when CMDaemon runs an action script |

All action scripts have the preceding standard environment variables set.

In `cmsh`, if the action object has its `node environment` parameter set to the value `yes`, then scripts running on a node are enabled with an extended environment that provides many more `CMD_*` environment variables. Otherwise they run in the standard environment. There are more environment variables besides the standard and node environment ones.

A list of the standard or extended environment variables can be dumped by running the system command `env > /tmp/dumpfile` within an action script, such as the test example script, and triggering the script to run.

Many of the environment variables are similar to the ones used by initialize and finalize scripts (section E.3 of the *Administrator Manual*) in the node-installer environment.

### 3.3.2 Extended Environment Variables Available To Action Scripts

If the action object has its `node environment` parameter set to the value `yes`, then scripts run in an extended environment that provides many more `CMD_*` environment variables. Otherwise they run in the standard environment of section 3.3.1.

The following table shows the extended as well as the standard environment variables, with some example values:

*Table 3.3.2: Environment Variables For Nodes In The Extended Environment*

| Variable | Example Value |
|---|---|
| CMD_ACTION_NAME | myaction |
| CMD_ACTIVE_HEAD_NODE_IP | 10.141.255.254 |
| CMD_CLUSTERNAME | basecm10 |
| CMD_DEVICE_TYPE | HeadNode |
| CMD_ENTITY_NAME | basecm10 |
| CMD_ENTITY_TYPE | HeadNode |
| CMD_ENTITY_UUID | 78d29ab5-b415-486f-ad6a-04c5a983110f |
| CMD_ENVIRONMENT_CACHE_EPOCH_MILLISECONDS | 1733150703765 |
| CMD_ENVIRONMENT_CACHE_UPDATES | 15 |
| CMD_EXPORTS | /cm/node-installer@internalnet<br>/cm/node-installer/certificates@internalnet<br>/var/spool/burn@internalnet<br>/home@internalnet<br>/cm/shared@internalnet |
| CMD_FSEXPORT__SLASH_cm_SLASH_node_DASH_installer<*node-installer values*> | |

where <*node-installer values*> takes these substitutions:

| ***<node-installer values>*** | **example value** |
|---|---|
| _installer_AT_internalnet_ALLOWWRITE | false |
| _AT_internalnet_HOSTS | 10.141.0.0/16 |
| _AT_internalnet_PATH | /cm/node-installer |
| _SLASH_certificates_AT_internalnet_ALLOWWRITE | true |
| _SLASH_certificates_AT_internalnet_HOSTS | 10.141.0.0/16 |
| _SLASH_certificates_AT_internalnet_PATH | /cm/node-installer/certificates |

CMD_FSEXPORT__SLASH_<*path values*>

where <*path values*> takes these substitutions:

| ***<path values>*** | **example value** |
|---|---|
| cm_SLASH_shared_AT_internalnet_ALLOWWRITE | true |
| cm_SLASH_shared_AT_internalnet_HOSTS | 10.141.0.0/16 |
| cm_SLASH_shared_AT_internalnet_PATH | /cm/shared |
| home_AT_internalnet_ALLOWWRITE | true |
| home_AT_internalnet_HOSTS | 10.141.0.0/16 |
| home_AT_internalnet_PATH | /home |
| var_SLASH_spool_SLASH_burn_AT_internalnet_ALLOWWRITE | true |
| var_SLASH_spool_SLASH_burn_AT_internalnet_HOSTS | 10.141.0.0/16 |
| var_SLASH_spool_SLASH_burn_AT_internalnet_PATH | /var/spool/burn |

| | |
|---|---|
| CMD_HOSTNAME | basecm10 |
| CMD_HTTP_PORT | 8080 |
| CMD_INFO_MESSAGE | |

*...continues*

*Table 3.3.2: Environment Variables For Nodes In The Extended Environment ...continued*

| Variable | Example Value |
|---|---|
| The interfaces eth0 and eth1 that make up the environment variable names listed next are for a Type 1 network with ethernet interfaces. The environment variable names change if the interface names change. For example, with InfiniBand interfaces, the eth0, eth1 strings become ib0, ib1. | |
| CMD_INTERFACE_eth0_IP | 10.141.255.254 |
| CMD_INTERFACE_eth0_MAC | 00:00:00:00:00:00 |
| CMD_INTERFACE_eth0_MASK_BITS | 16 |
| CMD_INTERFACE_eth0_MTU | 1500 |
| CMD_INTERFACE_eth0_SPEED | |
| CMD_INTERFACE_eth0_STARTIF | ALWAYS |
| CMD_INTERFACE_eth0_TYPE | NetworkPhysicalInterface |
| CMD_INTERFACE_eth1_GATEWAY | 192.168.255.254 |
| CMD_INTERFACE_eth1_IP | 192.168.244.169 |
| CMD_INTERFACE_eth1_MAC | 00:00:00:00:00:00 |
| CMD_INTERFACE_eth1_MASK_BITS | 20 |
| CMD_INTERFACE_eth1_MTU | 1500 |
| CMD_INTERFACE_eth1_SPEED | |
| CMD_INTERFACE_eth1_STARTIF | ALWAYS |
| CMD_INTERFACE_eth1_TYPE | NetworkPhysicalInterface |
| CMD_INTERFACES | eth0 eth1 |
| CMD_IP | 10.141.255.254 |
| CMD_JSON_DATA | no |
| CMD_MAC | FA:16:3E:33:8E:10 |
| CMD_MANAGEMENT_IP | 10.141.255.254 |
| CMD_MEASURABLE_NAME | PageOut |
| CMD_MEASURABLE_PARAMETER | |
| CMD_MEASURABLE_TYPE | MonitoringMeasurableMetric |
| CMD_MEASURABLE_UUID | 24321ec1-275c-4fd2-92eb-2ad7494216a8 |

...continues

*Table 3.3.2: Environment Variables For Nodes In The Extended Environment ...continued*

| Variable | Example Value |
|---|---|
| CMD_MULTI_ACTION | no |
| CMD_MYSQL_SOCKET | /var/run/mysqld/mysqld.sock |
| CMD_NODEGROUPS | |
| CMD_NODE_INSTALLER_PATH | /cm/node-installer |
| CMD_PARTITION | base |
| CMD_PASSIVE_HEAD_NODE_IP | |
| CMD_PORT | 8081 |
| CMD_PRIMARY_HEAD_NODE | basecm10 |
| CMD_PRODUCER_NAME | ProcVMStat |
| CMD_PROTOCOL | https |
| CMD_RAW_VALUE | 103603.200000 |
| CMD_ROLES | SlurmServer Storage Backup Boot Provisioning Monitoring SlurmAccounting Firewall HeadNode SlurmSubmit |
| CMD_SCRIPTTIMEOUT | 5 |
| CMD_SEVERITY | 10 |
| CMD_SHARED_HEAD_NODE_IP | |
| CMD_STATE_FLAPPING | no |
| CMD_STATUS_BURNING | NO |
| CMD_STATUS_CLOSED | NO |
| CMD_STATUS_HEALTHCHECK_FAILED | YES |
| CMD_STATUS_HEALTHCHECK_UNKNOWN | NO |
| CMD_STATUS_MESSAGE | |
| CMD_STATUS_MUTED | NO |
| CMD_STATUS_RESTART_REQUIRED | NO |
| CMD_STATUS_STATEFLAPPING | NO |
| CMD_STATUS_TERMINATED | NO |
| CMD_STATUS_TOOLMESSAGE | |
| CMD_STATUS | UP |
| CMD_STATUS_UPDATE_INDEX | 4 |

...continues

*Table 3.3.2: Environment Variables For Nodes In The Extended Environment ...continued*

| Variable | Example Value |
|---|---|
| CMD_STATUS_USERMESSAGE | |
| CMD_SYSINFO_SYSTEM_MANUFACTURER | OpenStack Foundation |
| CMD_SYSINFO_SYSTEM_NAME | OpenStack Nova |
| CMD_TRIGGER_EXPRESSION | (basecm10, *, *) > 50 |
| CMD_TRIGGER_NAME | killallyestrigger |
| CMD_TRIGGER_UUID | 1b784150-3f4f-4cb2-af37-993bbbe3dea4 |
| CMD_UPDATE_EPOCH_MILLISECONDS | 1733150703765 |
| CMD_USERDEFINED1 | |
| CMD_USERDEFINED2 | |
| CMD_UUID | 78d29ab5-b415-486f-ad6a-04c5a983110f |
| CMD_VALUE | 101 KiB/s |
| CMD_VALUE_COUNT | 1 |
| CMD_VALUE_EVAL | true |
| CMD_VALUE_TIME | 1733151851527 |

### 3.3.3 Environment Variables Useful For Debugging

The following environmental variables can be handy for debugging:

*Table 3.3.2: Environment Variables For Nodes In The Extended Environment*

| Variable | Description |
|---|---|
| CMD_INFO_FD | Set file descriptor used by CMDaemon for collecting information (default value: 3) |
| CMD_DEBUG | Setting this to a value of 1 often gives more debug output (default value: 0) |

Examples of their use are given in sections 2.3, 2.4, and 2.13.

# 4

# CMDaemon REST API

Some data from CMDaemon can be accessed via its REST API.

The REST API typically allows data only to be retrieved for most calls. Exceptions are:

- the `Status` call, which can generate status notifications (section 4.2.1), and
- the `Event` call, which can generate events (section 4.2.9),

and which can take POST input to specify their calls.

## 4.1 Authentication, And Definition Of *<curlauth>*

Two forms of authentication are supported:

- Basic: HTTP authentication (`--basic` option of `curl`)
- Certificate: Certificate-based authentication (`--cert` option of `curl`). Certificate-based authentication is covered in section 6.4.2 of the *Administrator Manual*.

The following two commands give identical results:

```
[alice@basecm10 ~]$ curl -k --basic --user "alice:password" "https://master:8081/rest"
[alice@basecm10 ~]$ curl --cert ~/.cm/cert.pem --key ~/.cm/cert.key -k "https://master:8081/rest"
```

For security, it is best to use the certificate key-based version.

For convenience, the command and authority parts of the preceding two commands—that is the string in the line that includes the text from `curl` to `8081` in the two `curl` commands—is designated by <*curlauth*> in this chapter. Thus, each of the commands can be represented by:

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest"
```

This allows the reader to focus on the path segment and variables part of the API.

## 4.2 Browsing The API

A summary diagram of the REST API is shown in figure 4.1:

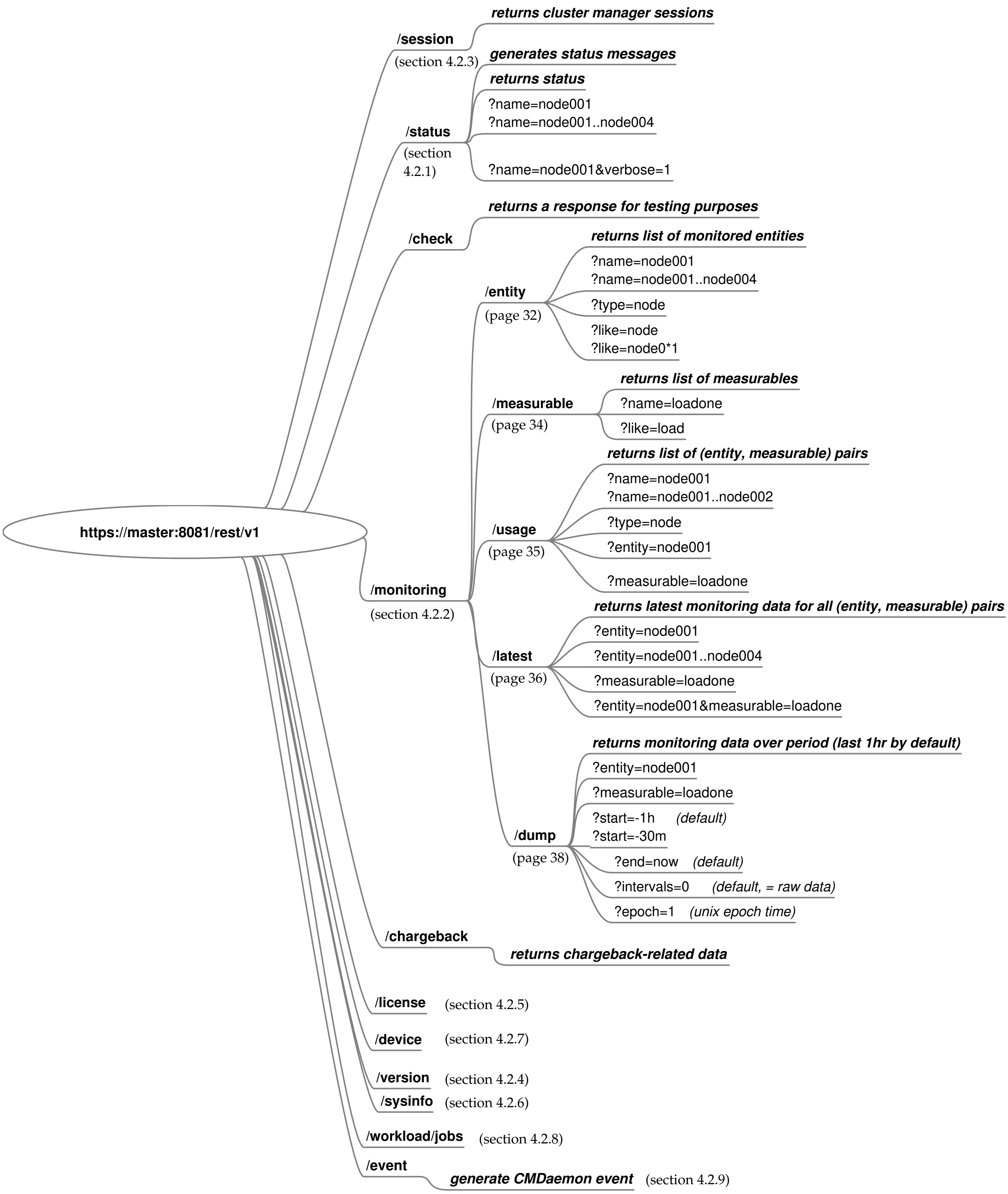


Figure 4.1: REST API summary tree

The remainder of this section elaborates upon the diagram.
The API directory structure is documented within the directory itself.

A GET operation on the main `/rest` entry point can list all subdirectories:

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest"
["v1"]
```

New lines are not part of the output by default. Setting a parameter of `1`, `2`, or more, for the `indent` variable uses newlines and an indentation of one, two, or more spaces. This makes the API output more readable for all API resource paths:

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest?indent=1"
[
 "v1"
]
```

Appending `/v1` to the URL gives the functionality available in the first version of the REST API.

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1?indent=1"
[
 "monitoring",
 "chargeback",
 "status",
 "session",
 "check",
 "version",
 "license",
 "sysinfo",
 "device",
 "workload",
 "event"
]
```

Appending `/monitoring` to the URL lists the subdirectory functionality available for monitoring.

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring?indent=1"
[
 "entity",
 "measurable",
 "latest",
 "dump",
 "usage"
]
```

### 4.2.1 Returning A Status, Or Generating A Status Message, Using `/v1/status`

**Returning A Status Using `/v1/status`**

The `status` resource path returns the UP/DOWN status for all devices:

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/status?indent=2"
[
  {
    "hostname": "basecm10",
    "status": "UP"
  },
  {
    "hostname": "node001",
    "status": "UP"
  },
  {
    "hostname": "node002",
    "status": "DOWN"
  }
]
```

The status can also be requested for a single device:

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/status?name=node001&indent=2"
[
  {
    "hostname": "node001",
    "status": "UP"
  }
]
```

The “two dots” list specification format (section 2.5.5 of the *Administrator Manual*) used in Base View and `cmsh` can also be used in the API:

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/status?name=node001..node002&indent=2"
[
  {
    "hostname": "node001",
    "status": "UP"
  },
  {
    "hostname": "node002",
    "status": "DOWN"
  }
]
```

For more detailed information, the verbose parameter can be added (output truncated):

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/status?verbose=1&indent=2"
[
  {
    "health_check_failed": true,
    "health_check_unknown": false,
    "hostname": "basecm10",
    "provisioning_failed": false,
    "restart_required": false,
```

```
    "status": "UP"
  },
  {
    "health_check_failed": true,
    "health_check_unknown": false,
    "hostname": "node001",
    "provisioning_failed": false,
...
```

#### Generating A Status Message Using `/v1/status`

A message can be associated with the status. An input can be as in the following `rest.in` file:

**Example**

```
[alice@basecm10 ~]$ cat /tmp/rest.in
[
  {
    "hostname": "node001",
    "user": "user message1",
    "info": "info message1",
    "tool": tool message1"
  },
  {
    "hostname": "node002",
    "user": "user message2",
    "info": "info message2",
    "tool": tool message2"
  }
]
```

The usual `curl` authentication string used so far, <*curlauth*> (section 4.1) is slightly modified from its value of:

```
curl --cert ~/.cm/cert.pem --key ~/.cm/cert.key -k "https://master:8081
```

to

```
curl --cert ~/.cm/cert.pem --key ~/.cm/cert.key -k --data "@/tmp/rest.in" "https://master:8081
```

This modified version allows POST data to be entered. The modified version can be called <*curlauthpost*>, and can be used as follows, returning a vector with the components having an integer value of 0, up to 3:

**Example**

```
[alice@basecm10 ~]$ <curlauthpost>/rest/v1/status"
[
0,
3
]
```

The dimension of the vector is in the current example is 2, and corresponds to the number of hostnames. Thus, the first component is associated with the hostname `node001`, and the second component is associated with hostname `node002`.

The value of each component returns the number of fields in the POST file that were modified by the API call. The fields that are evaluated are the optional fields `user`, `info`, and `tool`.

The event is logged in the event logger, by default at `/var/spool/cmd/events.log`, as:

**Example**

```
Mon Jun 17 18:15:50 2024 [notice] basecm10: node001, status: UP, reported: UP, time: 1718113341752,
  info message: info node1, user message: user node1, tool message: tool node1 (index: 27, display: 1)
Mon Jun 17 18:20:30 2024 [notice] basecm10: node002, status: UP, reported: UP, time: 1718113666566,
  info message: info node2, user message: user node2, tool message: tool node2 (index: 28, display: 1)
```

An entry is also made in `/var/log/cmdaemon`:

**Example**

```
Jun 17 18:51:57 basecm10 cmd[2586]: [   CMD   ]   Info: [Service::post_v1_status], update node001, changes: 1
Jun 17 18:51:57 basecm10 cmd[2586]: [   CMD   ]   Info: [Service::post_v1_status], update node002, changes: 1
```

With the default settings of `cmsh`, a window running `cmsh` shows:

**Example**

```
[root@basecm10 ~]# cmsh
[basecm10]%
Mon Jun 17 18:51:58 2024 [notice] basecm10: node001 [   UP    ] (info node1) (user node1) (tool  node1)
Mon Jun 17 18:51:58 2024 [notice] basecm10: node002 [   UP    ] (info node2) (user node2) (tool  node2)
```

Messaging via the REST API is somewhat similar to the event bucket InfoMessages feature (section 12.10.4 of the *Administrator Manual*) but developers should find the REST API version cleaner.

### 4.2.2 Monitoring Using `/v1/monitoring`

**Entities Via** `/v1/monitoring/entity`

The `entity` resource returns information about the entities that are known to the monitoring system. It is possible for an entity known to the monitoring system to have no data.

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/entity?indent=1"
{
 "entities": [
  {
   "key": 12884901889,
   "name": "default",
   "type": "Category"
  },
  {
   "key": 17179869185,
   "name": "globalnet",
   "type": "Network"
  },
  {
   "key": 17179869186,
   "name": "internalnet",
   "type": "Network"
  },
...
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/entity?name=node001&indent=1"
{
 "entities": [
  {
   "key": 38654705666,
```

```
   "name": "node001",
   "type": "PhysicalNode"
  }
 ]
}
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/entity?type=node&indent=1"
{
 "entities": [
  {
   "key": 38654705665,
   "name": "basecm10",
   "type": "HeadNode"
  },
  {
   "key": 38654705666,
   "name": "node001",
   "type": "PhysicalNode"
  },
  {
   "key": 38654705667,
   "name": "node002",
   "type": "PhysicalNode"
  }
 ]
}
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/entity?name=node001..node002&indent=1"
{
 "entities": [
  {
   "key": 38654705666,
   "name": "node001",
   "type": "PhysicalNode"
  },
  {
   "key": 38654705667,
   "name": "node002",
   "type": "PhysicalNode"
  }
 ]
}
```

A regex matcher can be used to find entities based on a name match:

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/entity?like=lobal&indent=1"
{
 "entities": [
  {
   "key": 17179869185,
   "name": "globalnet",
   "type": "Network"
  }
 ]
}
```

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/entity?like=nod.0*1&indent=1"
{
 "entities": [
  {
   "key": 38654705666,
   "name": "node001",
   "type": "PhysicalNode"
  }
 ]
}
```

Regexes are based on the ECMAScript specification (`https://en.cppreference.com/w/cpp/regex/ecmascript`).

**Measurables Via** `/v1/monitoring/measurable`
This entry returns information about the defined measurables.

### Example

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/measurable?indent=1"
{
 "measurables": [
  {
   "key": 261993005057,
   "name": "IpForwDatagrams",
   "type": "metric"
  },
  {
   "key": 261993005058,
   "name": "IpFragCreates",
   "type": "metric"
  },
  {
   "key": 261993005059,
   "name": "IpFragFails",
   "type": "metric"
  },
...typically hundreds more lines...

[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/measurable?name=loadone&indent=1"
{
 "measurables": [
  {
   "key": 261993005138,
   "name": "LoadOne",
   "type": "metric"
  }
 ]
}
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/measurable?like=load&indent=1"
{
 "measurables": [
  {
   "key": 261993005136,
   "name": "LoadFifteen",
   "type": "metric"
```

```
  },
  {
   "key": 261993005137,
   "name": "LoadFive",
   "type": "metric"
  },
  {
   "key": 261993005138,
   "name": "LoadOne",
   "type": "metric"
  }
 ]
}
```


#### Data Usage Via `/v1/monitoring/usage`

The usage resource is intended to show which (entity, measurable) pairs have data. For example, nodes with only 1 disk do not have data, if their associated measurables have the string `sdb` in their name.

To get the complete usage:

### Example

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/usage?indent=1"
{
 "data": [
  {
   "entity": "default",
   "measurable": "CoresTotal"
  },
  {
   "entity": "default",
   "measurable": "CoresUp"
  },
```
*...typically hundreds more lines...*

It is also possible to get all the measurables for which a specific entity, such as `node001`, has data.

### Example

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/usage?entity=node001&indent=1"
{
 "data": [
  {
   "entity": "node001",
   "measurable": "IpForwDatagrams"
  },
  {
   "entity": "node001",
   "measurable": "IpFragCreates"
  },
  {
   "entity": "node001",
   "measurable": "IpFragFails"
  },
```
*...typically hundreds more lines...*

Or all entities which have data for a specific measurable such as `loadone`:

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/usage?measurable=loadone&indent=1"
{
 "data": [
  {
   "entity": "basecm10",
   "measurable": "LoadOne"
  },
  {
   "entity": "node001",
   "measurable": "LoadOne"
  }
 ]
}
```

#### The Latest Monitoring Data Via `/v1/monitoring/latest`

The `latest` resource can be used to retrieve the last known sampled data points.

It is possible to get the latest monitoring data for all (entity, measurable) pairs.

This may result in a lot of information: about 125 bytes per (entity, measurable) pair.

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/latest?indent=1"
{
 "data": [
  {
   "age": 47.868,
   "entity": "default",
   "measurable": "CoresTotal",
   "raw": 1.0,
   "time": 1540476088861,
   "value": "1"
  },
  {
   "age": 47.868,
   "entity": "default",
   "measurable": "CoresUp",
   "raw": 1.0,
   "time": 1540476088861,
   "value": "1"
  },
  {
   "age": 47.868,
   "entity": "default",
   "measurable": "NodesClosed",
   "raw": 0.0,
   "time": 1540476088861,
   "value": "0"
  },
  {
   "age": 47.868,
   "entity": "default",
   "measurable": "NodesDown",
   "raw": 0.0,
```

```
    "time": 1540476088861,
    "value": "0"
  },
```
*...typically thousands more lines...*

The latest data can be requested for a selection of entities and measurables.

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/latest?entity=node001&indent=1"
{
 "data": [
  {
   "age": 138.625,
   "entity": "node001",
   "measurable": "IpForwDatagrams",
   "raw": 0.0,
   "time": 1540476100389,
   "value": "0/s"
  },
  {
   "age": 138.625,
   "entity": "node001",
   "measurable": "IpFragCreates",
   "raw": 0.0,
   "time": 1540476100389,
   "value": "0/s"
  },
  {
   "age": 138.625,
   "entity": "node001",
   "measurable": "IpFragFails",
   "raw": 0.0,
   "time": 1540476100389,
   "value": "0/s"
  },
```
*...typically hundreds more lines...*

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/latest?entity=node001..node004&indent=1"
```
*...as for previous output but for the range of* `nodes001..node004...`

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/latest?measurable=LoadOne&indent=1"
{
 "data": [
  {
   "age": 114.099,
   "entity": "basecm10",
   "measurable": "LoadOne",
   "raw": 0.03,
   "time": 1540476351361,
   "value": "0.03"
  },
  {
   "age": 155.07,
   "entity": "node001",
```

```
    "measurable": "LoadOne",
    "raw": 0.0,
    "time": 1540476310390,
    "value": "0"
  }
 ]
}
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/latest?entity=node001&measurable=LoadOne&indent=1"
{
 "data": [
  {
    "age": 106.706,
    "entity": "node001",
    "measurable": "LoadOne",
    "raw": 0.0,
    "time": 1540476790390,
    "value": "0"
  }
 ]
}
```

**Historic Data Dump Via** `/v1/monitoring/dump`

Dumping historic data can be done using the entry point:

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/dump?<options>"
```

The `dump` resource has several options:

- `entity`: name or range of entities
- `measurable`: name of the measurable
- `start`: time to be plotted (default: `-1h`)
- `end`: end to be plotted (default: `now`)
- `intervals`: number of interpolation intervals (default: `0`, raw data)
- `epoch`: display timestamps as unix epoch (default: `0`)

The time specification format is the same one used for the `dumpmonitoringdata` command (section 12.6.4 of the *Administrator Manual*).

To prevent gigabytes of data being retrieved when no options are specified, `entity` and `measurable` must be specified.

If there is a need to dump all the monitoring data, then it can be done by specifying empty strings for both entity and measurable. For example, the following command dumps all raw data for the default last hour:

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/monitoring/dump?entity=&measurable=?indent=1"
{
 "data": [
  {
    "entity": "default",
    "measurable": "CoresTotal",
```

```
   "raw": 1.0,
   "time": "2018/10/25 13:15:28",
   "value": "1"
  },
  {
   "entity": "default",
   "measurable": "CoresTotal",
   "raw": 1.0,
   "time": "2018/10/25 16:35:28",
   "value": "1"
  },
  {
   "entity": "default",
   "measurable": "CoresUp",
   "raw": 1.0,
   "time": "2018/10/25 13:49:28",
   "value": "1"
  },
```

...*typically thousands more lines*...

### 4.2.3 Session Using `/v1/session`

The response to the `sessions` method is similar to the output from listing in `session` mode of `cmsh` (`cmsh -c "session list"`)

The method lists the sessions that the cluster manager is involved with.

#### Example

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/session?indent=1"
[
 {
  "address": "127.0.0.1",
  "group": "admin",
  "node": "basecm10",
  "type": "node",
  "username": ""
 },
 {
  "address": "10.141.255.254",
  "group": "admin",
  "type": "node",
  "username": ""
 },
 {
  "address": "10.141.0.1",
  "group": "node",
  "node": "node001",
  "type": "node",
  "username": ""
 },
 {
  "address": "10.141.0.2",
  "group": "node",
  "node": "node002",
  "type": "node",
  "username": ""
 }
```

### 4.2.4 Version Using `/v1/version`

The `version` method returns version parameters.

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/version?indent=1"
{
  "build_hash": "daf30669f1",
  "build_index": 152175,
  "cm_version": "9.2",
  "cmd_version": "2.2",
  "database_version": 36280
}
```

### 4.2.5 License Using `/v1/license`

The `license` method returns license parameters.

#### Example

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/license?indent=1"
{
  "acceleratorNodeCount": 0,
  "accountingAndReporting": true,
  "baseType": "LicenseInfo",
  "burstNodeCount": 0,
  "childType": "",
  "edgeSites": true,
  "edition": "Advanced",
  "endTime": 2177449140,
  "licenseType": "Commercial",
  "licensedAcceleratorNodes": 80,
  "licensedBurstNodes": 1000,
  "licensedNodes": 100,
  "licensee": "/C=US/ST=None/L=None/O=None/OU=None/CN=basecm10",
  "macAddress": "FA:16:3E:3B:94:98",
  "message": "",
  "modified": false,
  "nodeCount": 3,
  "oldLocalUniqueKey": 0,
  "refPartitionUniqueKey": 21474836481,
  "revision": "",
  "serial": 1017214,
  "startTime": 1508108400,
  "toBeRemoved": false,
  "uniqueKey": 281474976710653,
  "version": "7.0 and above"
}
```

### 4.2.6 Sysinfo Using `/v1/sysinfo`

The `sysinfo` method is similar to the `sysinfo` command in the `device` mode of `cmsh`. It returns information about some basic system hardware parameters.

#### Example

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/sysinfo?indent=1"
{
  "node001": {
    "baseType": "SysInfoCollector",
```

```
"biosDate": "04/01/2014",
"biosVendor": "SeaBIOS",
"biosVersion": "1.13.0-1ubuntu1.1",
"bootIf": "ens3",
"childType": "",
"clusterRandomNumber": 6332472641088672013,
"diskCount": 2,
"diskTotalSpace": 10745806848,
"disks": [
  {
    "baseType": "DiskInfo",
    "childType": "",
    "ioScheduler": "[mq-deadline] kyber bfq none",
    "model": "virtio",
    "modified": false,
    "name": "vda",
    "oldLocalUniqueKey": 0,
    "rev": "",
    "revision": "",
    "size": 8388608,
    "toBeRemoved": false,
    "uniqueKey": 281474976710948,
    "vendor": ""
  },
  {
    "baseType": "DiskInfo",
    "childType": "",
    "ioScheduler": "[mq-deadline] kyber bfq none",
    "model": "virtio",
    "modified": false,
    "name": "vdb",
    "oldLocalUniqueKey": 0,
    "rev": "",
    "revision": "",
    "size": 10737418240,
    "toBeRemoved": false,
    "uniqueKey": 281474976710949,
    "vendor": ""
  }
],
"extra": null,
"fabric": false,
"fips": false,
"fpgas": [],
"gpus": [],
"ibGUIDs": [],
"interconnects": [],
"memory": [
  {
    "IDs": [
      "0/0"
    ],
    "baseType": "MemoryInfo",
    "childType": "",
    "description": "DIMM RAM",
```

```
      "locations": [
        "DIMM 0"
      ],
      "modified": false,
      "oldLocalUniqueKey": 0,
      "revision": "",
      "size": 1073741824,
      "speed": 0,
      "toBeRemoved": false,
      "uniqueKey": 281474976710950
    }
  ],
  "memorySwap": 0,
  "memoryTotal": 1016152064,
  "modified": false,
  "motherboardManufacturer": "",
  "motherboardName": "",
  "nics": [
    "ens3"
  ],
  "oldLocalUniqueKey": 0,
  "osFlavor": "Rocky8u5",
  "osName": "Linux",
  "osVersion": "4.18.0-348.el8.0.2.x86_64",
  "parentUniqueKey": 85899345921,
  "processors": [
    {
      "IDs": [
        0
      ],
      "baseType": "Processor",
      "bogomips": 4190.15,
      "cacheSize": 16777216,
      "childType": "",
      "coreIDs": [
        0
      ],
      "cores": 1,
      "model": "Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz",
      "modified": false,
      "oldLocalUniqueKey": 0,
      "physicalIDs": [
        0
      ],
      "revision": "",
      "speed": 2095078000.0,
      "toBeRemoved": false,
      "uniqueKey": 281474976710947,
      "vendor": "GenuineIntel"
    }
  ],
  "raidControllers": [],
  "refDeviceUniqueKey": 38654705666,
  "revision": "",
  "seLinux": false,
```

```
    "systemManufacturer": "OpenStack Foundation",
    "systemName": "OpenStack Nova",
    "timestamp": 1651158566,
    "toBeRemoved": false,
    "uniqueKey": 85899345921,
    "updateCount": 5,
    "vendorTag": "5bf2a543-542d-4391-946c-abb648a09158",
    "virtualCluster": true
  },
  "node002": {
    "baseType": "SysInfoCollector",
    "biosDate": "04/01/2014",
...
}
```

### 4.2.7 Device Information Using `/v1/device`

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/device?indent=1"
[
  {
    "cluster": "basecm10",
    "hostname": "basecm10",
    "ip": "10.141.255.254",
    "mac": "FA:16:3E:EF:71:05",
    "network": "internalnet",
    "roles": [
      "backup",
      "storage",
      "firewall",
      "headnode",
      "monitoring",
      "provisioning",
      "boot"
    ],
    "type": "HeadNode"
  },
  {
    "category": "default",
    "cluster": "basecm10",
    "hostname": "node001",
    "ip": "10.141.0.1",
    "mac": "FA:16:3E:2B:A4:31",
    "network": "internalnet",
    "type": "PhysicalNode"
  },
  {
    "category": "default",
    "cluster": "basecm10",
    "hostname": "node002",
    "ip": "10.141.0.2",
    "mac": "FA:16:3E:D4:C8:5A",
    "network": "internalnet",
    "type": "PhysicalNode"
  }
]
```

### 4.2.8 WLM Information Using `/v1/workload`

The `workkoad` path has the following endpoints:

- `jobs`
- `drain`

These return information related to the endpoints.
The workload jobs path returns running jobs:

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/workload/jobs?indent=1"
[
 {
  "account": "projecty",
  "group": "alice",
  "job_id": "2301",
  "job_name": "iozone",
  "nodes": [
   "node001"
  ],
  "queue": "defq",
  "run_time": "4m 39s",
  "start_time": "2023/06/08 14:24:53",
  "state": "RUNNING",
  "sumbit_time": "2023/06/08 14:24:53",
  "user": "alice"
 },
 {
  "account": "projectx",
  "group": "charlie",
  "job_id": "2306",
  "job_name": "sleep",
  "nodes": [
   "node001"
  ],
  "queue": "defq",
  "run_time": "3m 34s",
  "start_time": "2023/06/08 14:25:58",
  "state": "RUNNING",
  "sumbit_time": "2023/06/08 14:25:57",
  "user": "charlie"
 },
 {
  "account": "projecty",
  "group": "alice",
  "job_id": "2307",
  "job_name": "iozone",
...
```

The workload drain path returns node drain status information:

**Example**

```
[alice@basecm10 ~]$ <curlauth>/rest/v1/workload/drain?indent=1"
[
```

```
 {
  "hostname": "node001",
  "queue": "defq",
  "reason": "",
  "status": "UNDRAINED"
 },
 {
  "hostname": "node002",
  "queue": "defq",
  "reason": "",
  "status": "UNDRAINED"
 },
 ...
```

### 4.2.9 Event Generation Using `/v1/event`

An event (section 12.10 of the *Administrator Manual*) can be generated in CMDaemon from a JSON format input used with the `event` endpoint.

An input can be as in the following `rest.in` file:

**Example**

```
[alice@basecm10 ~]$ cat /tmp/rest.in
{
  "message": "hello world",
  "details": "send via rest",
  "severity": "notice"
}
```

The usual `curl` authentication string used so far, <*curlauth*> (section 4.1) is slightly modified from its value of:

```
curl --cert ~/.cm/cert.pem --key ~/.cm/cert.key -k "https://master:8081
```

to

```
curl --cert ~/.cm/cert.pem --key ~/.cm/cert.key -k --data "@/tmp/rest.in" "https://master:8081
```

This modified version allows POST data to be entered. The modified version can be called <*curlauthpost*>, and can be used as follows, returning `true`:

**Example**

```
[alice@basecm10 ~]$ <curlauthpost>/rest/v1/event"
true
```

The event is logged in the event logger, by default at `/var/spool/cmd/events.log`, as:

**Example**

```
Tues Jun  4 11:09:21 2024 [notice] basecm10: hello world
send via rest
```

With the default settings of `cmsh`, a window running `cmsh` shows:

**Example**

```
[root@basecm10 ~]# cmsh
[basecm10]%
Tue Jun  4 11:09:21 2024 [notice] basecm10: hello world
For details type: events details 1
```

and, if as suggested, `events details 1` is typed, the value of `details` from the input is seen:

```
[basecm10]% events details 1
send via rest
```

# 5

# BCM JSON API

This chapter documents the JSON API services and entities available for NVIDIA Base Command Manager.

The BCM head node landing page (section 2.4.1 of the *Administrator Manual*) links via the `CM API Docs` tile (the second tile in figure 5.1) to the API reference documentation for all available services and entities:

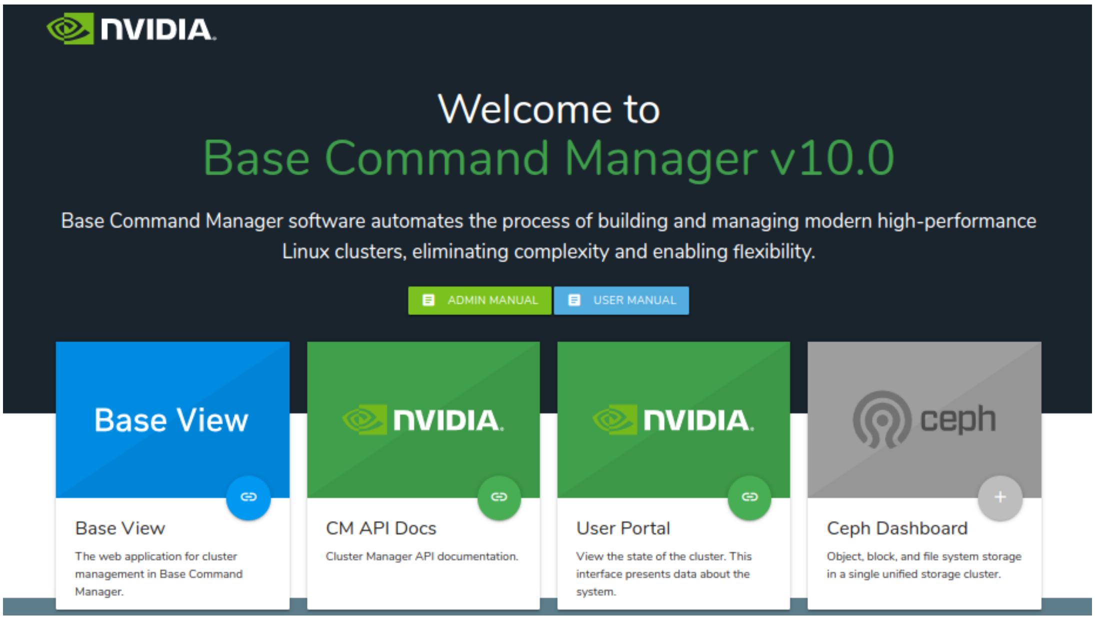


Figure 5.1: Head node hostname or IP address landing page at `https://`*<host name or IP address>*

It can also be accessed via the user portal of the cluster by clicking on the `JSON API documentation` link in the documentation section of the home page (Section 12.8.4 of the *Administrator Manual*).

By default, the direct API URL takes the form:

`https://`*<head node address name or IP address>*`:8081/api`

At that URL:

- the `Search` page can be used to list services, entities, events, and RPCs
- the `Inheritance` page can be used to display the entities hierarchy

Within the search page (figure 5.2),

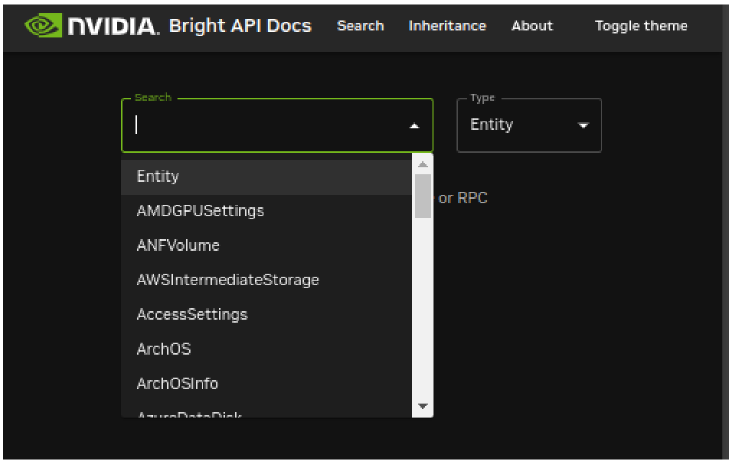


Figure 5.2: Search page for API documentation

- if the Type option is set to Service, then the drop-down list for Search presents the list of services
- if the Type option is set to Entity, then the drop-down list for Search presents the list of entities

## 5.1 API Services

If a service is selected from the drop-down list for Search, then its RPCs are displayed. Each RPC shows the tokens required for its use. Each RPC in turn can be expanded to display its request format (call and arguments) and response format (figure 5.3):

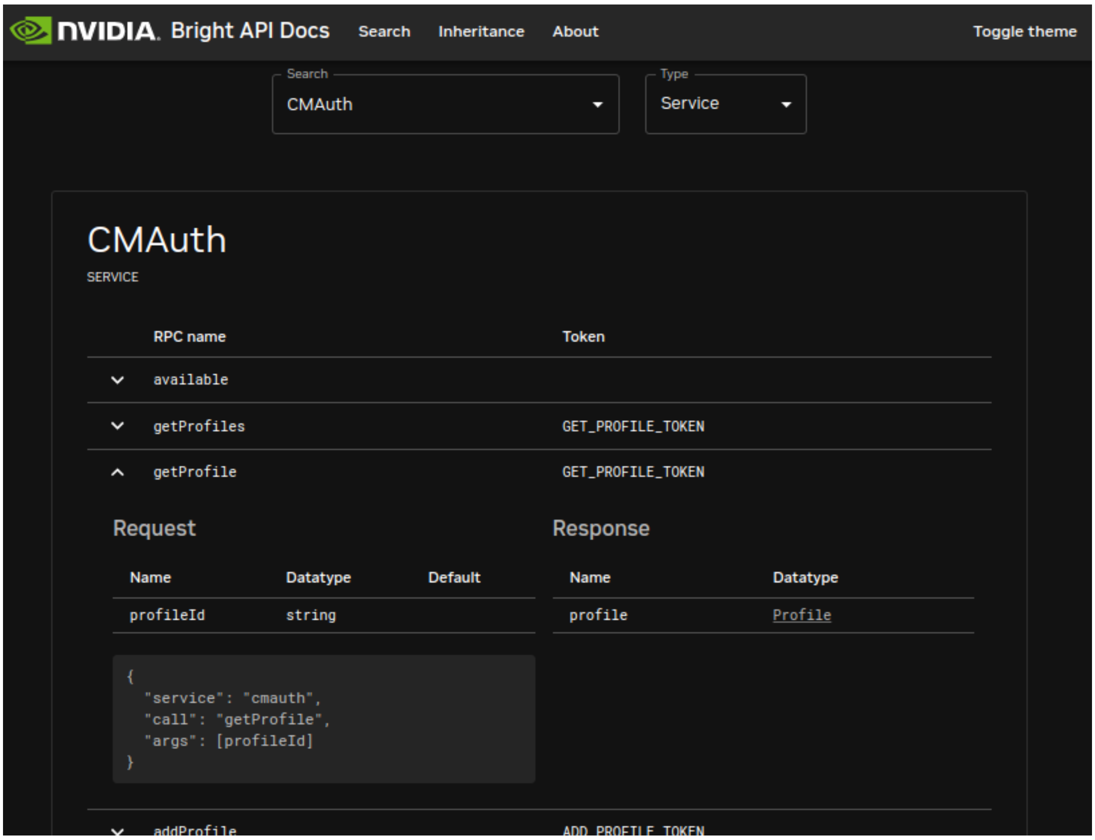


Figure 5.3: Example of an API documentation search page display result for the expanded view of the getProfile RPC of the CMAuth service

### 5.1.1 API Services List

The list of services are:

```
CMAuth
CMBeeGFS
CMCeph
CMCert
CMCloud
CMDevice
CMEtcd
CMGui
CMJob
CMKube
CMMain
CMMon
CMNet
CMPart
CMProc
CMProv
CMServ
CMSession
CMStatus
CMTest
CMUser
```

## 5.2 API Entities

If an entity is selected from the drop-down list for `Search`, then its properties are displayed. (figure 5.4):

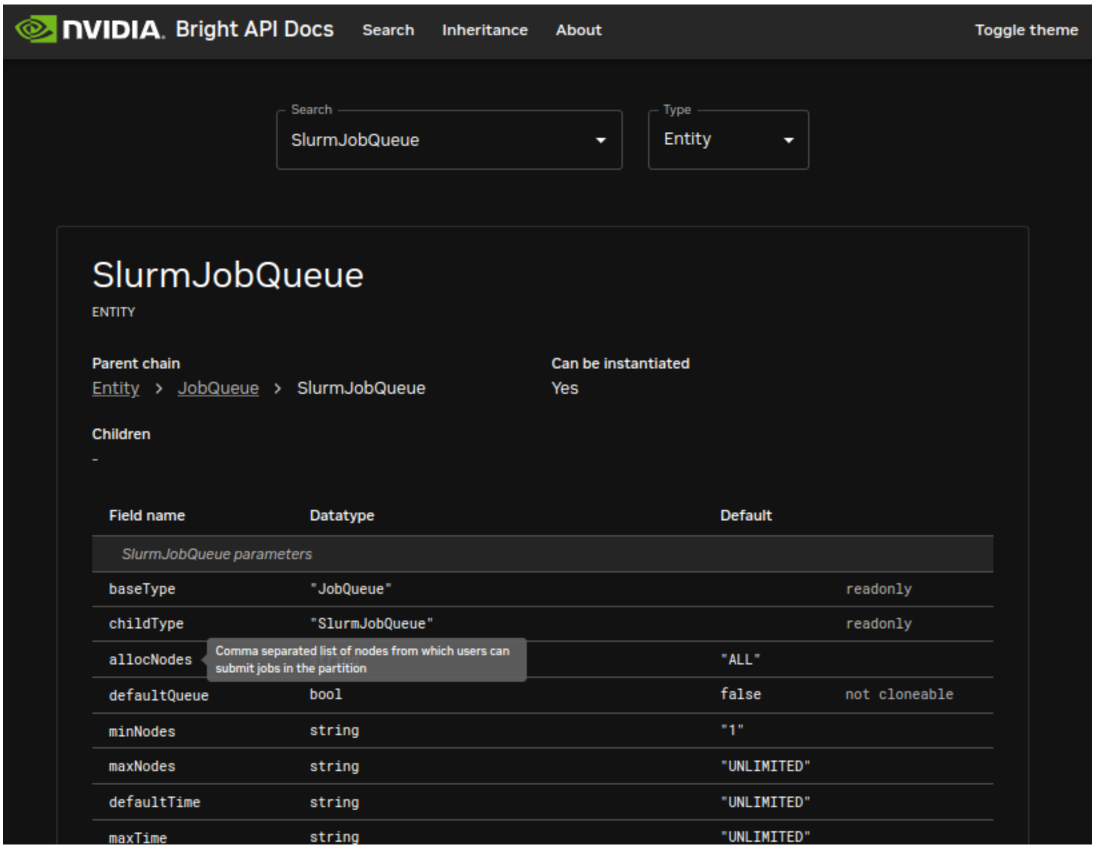


Figure 5.4: Example of an API documentation search page display result for the `SlurmJobQueue` entity

Each entity parameter typically has hovertext that describes it. For example, in figure 5.4 the terse `allocNodes` parameter of the `SlurmJobQueue` entity has a helpful associated hovertext description of `Comma-separated list of nodes from which users can submit jobs into the system.`

### 5.2.1 API Entities List

The list of API entities can be viewed in the search page display (figure 5.2).

By default the `Inheritance` page for API entities is located at

`https://`<*head node address name or IP address*>`:8081/api/inheritance`

The list of API entities can also conveniently be viewed there in a hierarchy:

```
Entity
|-- ANFVolume
|-- AccessSettings
|-- ArchOSInfo
|   '-- ArchOS
|-- AzureDisk
|   |-- AzureDataDisk
|   '-- AzureOSDisk
...
```

## 5.3 JSON Examples

**complete.sh**

```
#!/bin/bash
```

```
URL=https://localhost:8081/json/
user=root
pass=secretrootpassword

echo "========== login ==========="
curl -c curl.cookiest.txt -i -k -X POST -d \
'{"service":"login", "username":"root", "password":"'$pass'"}' $URL; echo

echo "========= master =========="
curl --cookie curl.cookiest.txt -i -k -X POST -d \
'{"service":"cmdevice","call":"getNode", "arg":"master"}' $URL; echo

echo "========== logout =========="
curl --cookie curl.cookiest.txt -i -k -X POST -d \
'{"service":"logout"}' $URL; echo

echo "========== denied =========="
curl --cookie curl.cookiest.txt -i -k -X POST -d \
'{"service":"cmdevice","call":"getNode", "arg":"master"}' $URL; echo
rm -f curl.cookiest.txt

echo "=========== cert =========="
curl --cert $HOME/.cm/admin.pem --key $HOME/.cm/admin.key -i -k -X POST -d \
'{"service":"cmdevice","call":"getNode","arg":"master"}' $URL; echo
```

### curl.sh

```
#!/bin/bash

URL=https://localhost:8081/json/

if [ -z "$1" ]; then
  read -p "pass: " -s pass
else
  pass=$1
fi

curl -c curl.cookiest.txt -i -k -X POST -d \
'{"service":"login", "username":"root", "password":"'$pass'"}' $URL

# curl --cookie curl.cookiest.txt -i -k -X POST -d \
'{"service":"cmsession","call":"getLastEvents","args":[0,256]}' $URL

curl --cookie curl.cookiest.txt -i -k -X POST -d \
'{"service":"cmmain","call":"getProfile"}' $URL

curl --cookie curl.cookiest.txt -i -k -X POST -d \
'{"service":"cmmain","call":"getSubjectName"}' $URL
```

### devices.sh

```
#!/bin/bash
URL=https://localhost:8081/json/
```

```
if [ "$1" == "gzip" ]; then
   wget --certificate=$HOME/.cm/admin.pem --private-key=$HOME/.cm/admin.key \
   --header='Accept-Encoding: gzip' \
   --no-check-certificate --server-response -qO- $URL \
   --post-data='{"service":"cmdevice","call":"getDevices"}'
else
   wget --certificate=$HOME/.cm/admin.pem --private-key=$HOME/.cm/admin.key --no-check-certificate \
   --server-response -qO- $URL --post-data='{"service":"cmdevice","call":"getDevices"}'
fi
```

Tip: run as `./devices.sh | python -mjson.tool`.

#### loadone.sh

```
#!/bin/bash

URL=https://localhost:8081/json/
# not perfect but gets the job done
function jsonval {
    temp=$(echo $json | sed 's/\\\\\//\//g' | sed 's/[{}]//g' |
        awk -v k="text" '{n=split($0,a,","); for (i=1; i<=n; i++) print a[i]}' |
        sed 's/\"\:\"/\|/g' | sed 's/[\,]/ /g' | sed 's/\"//g' | grep -w $prop)

    r=$(echo ${temp##*|} | tr ']' ' ' | tr ' ' '\n' | cut -d: -f2 | sort -n)
    echo $(echo $r | cut -d' ' -f 1)
}

prop='uuid'
node=master
json=$(2>/dev/null wget --certificate=$HOME/.cm/admin.pem \
    --private-key=$HOME/.cm/admin.key \
    --no-check-certificate \
    --server-response \
    -qO- $URL \
    --post-data='{"service":"cmdevice","call":"getDevice","arg":"'$node'"}')
nkey=$(jsonval)

if [ -z "$nkey" ]; then
    echo "$json"
    exit 1
fi

echo "$node.uuid = $nkey"
json=$(2>/dev/null wget --certificate=$HOME/.cm/admin.pem \
    --private-key=$HOME/.cm/admin.key \
    --no-check-certificate \
    --server-response \
    -qO- $URL \
    --post-data='{"service":"cmmon","call":"getMonitoringMeasurable","name":"LoadOne"}')
mkey=$(jsonval)
echo "loadone.uuid = $mkey"
now=$(date +%s)
day=$((now-86400))
echo "now is $now"
echo "day is $day"
```

```
cat <<EOF > /tmp/plot.json
{ "service" : "cmmon",
"call" : "plot",
"request" : { "entities" : ["$nkey"],
"measurables" : ["$mkey"],
"intervals" : 25,
"rangeStart" : $((day*1000)),
"rangeEnd" : $((now*1000))
}
}
EOF

2>/dev/null wget --certificate=$HOME/.cm/admin.pem \
    --private-key=$HOME/.cm/admin.key \
    --no-check-certificate \
    -qO- $URL \
    --post-file=/tmp/plot.json | \
    python -mjson.tool
```

### login.sh

```
#!/bin/bash
URL=https://localhost:8081/json/
user=$USER
pass=secretpassword
wget --keep-session-cookies --save-cookies cookie.txt --no-check-certificate \
--server-response -qO- $URL --post-data='{"service":"login","username":"'$user'","password":"'$pass'"}'
echo
```

### logout.sh

```
#!/bin/bash
URL=https://localhost:8081/json/
wget --load-cookies cookie.txt --no-check-certificate --server-response -qO- $URL \
--post-data='{"service":"logout"}'
rm cookie.txt
echo
```

### node001.sh

```
#!/bin/bash
URL=https://localhost:8081/json/

if [ -z "$1" ]; then
  node=node001
else
  node=$1
fi

wget --certificate=$HOME/.cm/admin.pem --private-key=$HOME/.cm/admin.key \
--no-check-certificate --server-response -qO- $URL \
--post-data='{"service":"cmdevice","call":"getDevice","arg":"'$node'"}' | python -mjson.tool
```

#### basic_information.sh

```
#!/bin/bash
URL=https://localhost:8081/json/
wget --certificate=$HOME/.cm/admin.pem --private-key=$HOME/.cm/admin.key \
--no-check-certificate --server-response -qO- $URL \
--post-data='{"service":"cmpart","call":"getBasicEntityInformation"}'
```

#### push_to_CMDaemon.sh

In the following example, the health check `ManagedServicesOK`, is pushed to CMDaemon with a `FAIL` value.

**Example**

```
[root@basecm10 ~]# cat push_to_CMDaemon.sh
#!/bin/bash
URL='https://master:8081/monitoring/push/ManagedServicesOk?info=brol&class=Push/Single&healthcheck=yes'
value='FAIL'
curl --cert $HOME/.cm/admin.pem --key $HOME/.cm/admin.key -i -k -X POST -d "$value" $URL; echo
```

Its behavior can be verified by checking the latest value for `ManagedServicesOK` before and after the `push_to_CMDaemon.sh` script is run:

**Example**

```
[root@basecm10 ~]# curl --cert ~/.cm/admin.pem --key ~/.cm/admin.key -k
"https://master:8081/rest/v1/monitoring/latest?measurable=ManagedServicesOK&entity=basecm10&indent=1"
{
 "data": [
  {
   "age": 89.735,
   "entity": "basecm10",
   "measurable": "ManagedServicesOk",
   "raw": 0.0,
   "time": 1586450030968,
   "value": "PASS"
  }
 ]
}

[root@basecm10 ~]#./push_to_CMDaemon.sh
HTTP/1.1 200 OK
Content-Length: 55
Content-Type: application/json

{
  "values": {
    "added": 1,
    "provided": 1
  }
}
[root@basecm10 ~]# curl --cert ~/.cm/admin.pem --key ~/.cm/admin.key -k
"https://master:8081/rest/v1/monitoring/latest?measurable=ManagedServicesOK&entity=basecm10&indent=1"
{
 "data": [
  {
```

```
    "age": 3.357,
    "entity": "basecm10",
    "info": "brol",
    "measurable": "ManagedServicesOk",
    "raw": 2.0,
    "time": 1586450124437,
    "value": "FAIL"
   }
 ]
}
```


A metric version of the push, using the measurable `push-test-02` might look like:

```
#!/bin/bash
URL='https://localhost:8081/monitoring/push/push-test-02?info=brol&class=Push/Single&unit=s'
value=$(date +%s)
curl --cert $HOME/.cm/admin.pem --key $HOME/.cm/admin.key -i -k -X POST -d "$value" $URL; echo
```

A collection can be pushed as follows: To initialize (once):

```
#!/bin/bash
URL='https://localhost:8081/monitoring/initialize'
curl --cert $HOME/.cm/admin.pem --key $HOME/.cm/admin.key -i -k -X POST -d \
'[
   {"metric":"push-collection-01","class":"Push/Collection"},
   {"metric":"push-collection-02","class":"Push/Collection"}
 ]' $URL; echo
```

After initializing, sampling can be done with:

```
#!/bin/bash
URL='https://localhost:8081/monitoring/push'
curl --cert $HOME/.cm/admin.pem --key $HOME/.cm/admin.key -i -k -X POST -d \
'[
   {"metric":"push-collection-01","value":31},
   {"metric":"push-collection-02","value":32,"info":"Some message"}
 ]' $URL; echo
```