Bright Cluster Manager 8.1

# Developer Manual

Revision: c2da708

Date: Mon Aug 17 2020

## 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 Novell, Inc. PGI is a registered trademark of NVIDIA Corporation. FLEXlm is a registered trademark of Flexera Software, Inc. PBS Professional, PBS Pro, 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. Bright Computing, Inc. shall not be liable for technical or editorial errors or omissions which may occur in this document. Bright Computing, Inc. shall not be liable for any damages resulting from the use of this document.

## Limitation of Liability and Damages Pertaining to Bright Computing, Inc.

The Bright Cluster Manager product principally consists of free software that is licensed by the Linux authors free of charge. Bright Computing, Inc. shall have no liability nor will Bright Computing, Inc. provide any warranty for the Bright Cluster 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 Bright Cluster 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 Bright Cluster 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 Bright Cluster Manager 8.1.

## 0.1 About This Manual

This manual is aimed at helping developers who would like to program the Bright Cluster 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 `cmgui` 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

Regularly updated versions of the Bright Cluster Manager 8.1 manuals are available on updated clusters by default at `/cm/shared/docs/cm`. The latest updates are always online at `http://support.brightcomputing.com/manuals`.

- 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 Bright Cluster Manager.
- The *OpenStack Deployment Manual* describes how to deploy OpenStack with Bright Cluster Manager.
- The *Big Data Deployment Manual* describes how to deploy Big Data with Bright Cluster Manager.
- The *Machine Learning Manual* describes how to install and configure machine learning capabilities with Bright Cluster Manager.

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 Bright Cluster Manager environment and the addition of new hardware and/or applications. The manuals also regularly incorporate customer feedback. Administrator and user input is greatly valued at Bright Computing. So any comments, suggestions or corrections will be very gratefully accepted at `manuals@brightcomputing.com`.

There is also a feedback form available via Bright View, via the Account icon,  , following the clickpath:

`Account`→`Help`→`Feedback`

## 0.3 Getting Administrator-Level Support

If the reseller from whom Bright Cluster Manager was bought offers direct support, then the reseller should be contacted.

Otherwise the primary means of support is via the website `https://support.brightcomputing.com`. This allows the administrator to submit a support request via a web form, and opens up a trouble ticket. It is a good idea to try to use a clear subject header, since that is used as part of a reference tag as the ticket progresses. Also helpful is a good description of the issue. The followup communication for this ticket goes via standard e-mail. Section 13.2 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, Bright Computing can be contacted in order to arrange a support contract.

## 0.5 Getting Professional Services

Bright Computing normally differentiates between professional services (customer asks Bright Computing to do something or asks Bright Computing to provide some service) and support (customer has a question or problem that requires an answer or resolution). Professional services can be provided after consulting with the reseller, or the Bright account manager.

# 1

# Bright Cluster Manager Python API

This chapter introduces the Python API of Bright Cluster Manager. For a head node `bright81`, the API reference documentation for all available objects is available in a default cluster via browser access to the URL:

```
https://bright81/userportal/downloads/python
```

The preceding access is via the User Portal (section 12.7 of the *Administrator Manual*).

The documentation is also available directly on the head node itself at:

```
file:///cm/local/docs/cmd/python/index.html
```

## 1.1 Installation

The Python cluster manager bindings are pre-installed on the head node.

### 1.1.1 Linux Clients

For Linux clients, a redistributable source package is supplied in the `pythoncm-dist` package installed on the cluster. The file at `/cm/shared/apps/pythoncm/dist/pythoncm-8.1-r18836-src.tar.bz2`—the exact version number may differ—is copied and untarred to any directory.

The `build.sh` script is then run to compile the source. About 4GB of memory is usually needed for compilation, and additional packages may be required for compilation to succeed. A list of packages needed to build Python cluster manager bindings can be found in the `README` file included with the package.

The `headnodeinfo.py` example supplied with the untarred files is edited as for in the earlier windows client example, for the `clustermanager.addCluster` line.

The path to the remote cluster manager library is added:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:remotecm
```

To verify everything is working, the following can be run:

```
python ./headnodeinfo.py
```

## 1.2 Examples

A set of examples can be found in `/cm/local/examples/cmd/python/` on the head node of the cluster.

### 1.2.1 First Program

A Python script is told to use the cluster manager bindings by importing `pythoncm` at the start of the script:

```
import pythoncm
```

If not working on the cluster, the administrator needs to set the path where the shared libraries can be found (`pythoncm.so` in Linux, or `python.pyd` in windows). This is done by adding the following to the start of the script:

```
import sys
sys.path.append(".")     # path to pythoncm.so/python.pyd
```

Python cluster manager bindings allow for simultaneous connections to several clusters. For this reason the first thing to do is to create a ClusterManager object:

```
clustermanager = pythoncm.ClusterManager()
```

A connection to a cluster can now be made. There are two possible ways of connecting.

The first is using the certificate and private key file that `cmsh` uses by default when it authenticates from the head node.

```
cluster = clustermanager.addCluster('https://mycluster:8081',\
'/root/.cm/admin.pem', '/root/.cm/admin.key');
```

The second way uses the password protected `admin.pfx` file, which is generated with the `cmd -c` command. A Python script could ask for the password and store it in a variable for increased security.

```
cluster = clustermanager.addCluster('https://mycluster:8081',\
'/root/.cm/cmgui/admin.pfx', '', '<password>');
```

Having defined the cluster, a connection can now be made to it:

```
isconnected = cluster.connect()
if !isconnected:
 print "Unable to connect"
 print cluster.getLastError()
 exit(1)
```

If a connection cannot be made, the function `cluster.connect()` returns false. The function `cluster.getLastError()` shows details about the problem. The two most likely problems are due to a wrong password setting or a firewall settings issue.

Similar to `cmgui` and `cmsh`, the cluster object contains a local cache of all objects. This cache will be filled automatically when the connection is established. All changes to properties will be done on these local copies and will be lost after the Python scripts exits, unless a `commit` operation is done.

The most common operation is finding specific objects in the cluster.

```
active = cluster.find('active')
if active == None:
 print "Unable to find active head node"
 exit(1)
else:
 print "Hostname of the active head node is %s" % active.hostname
```

If creating an automated script that runs at certain times, then it is highly recommended to check if objects can be found. During a failover, for instance, there will be a period over a few minutes in which the active head node will not be set.

It is good practice to disconnect from the cluster at the end of the script.

```
cluster.disconnect()
```

When connecting to a cluster with a failover setup, it is the shared IP address that should be connected to, and not the fixed IP address of either of the head nodes.

## 1.3 Methods And Properties

### 1.3.1 Viewing All Properties And Methods

All properties visible in `cmsh` and `cmgui` are also accessible from Python cluster manager bindings. The easiest way to get an overview of the methods and properties of an object is to define the following function:

```
import re
def dump(obj):
  print "--- DUMP ---"
  for attr in dir(obj):
    p = re.compile('^__.*__$')
    if not p.match(attr):
      print "%s = %s" % (attr, getattr(obj, attr))
```

An overview of all properties and methods for the active head node can be obtained with:

```
active = cluster.find('active')
dump(active)
```

### 1.3.2 Property Lists

Most properties are straightforward and their names are almost identical to the `cmsh` equivalent.

For instance:

```
node.mac = '00:00:00:00:00:00'
category.softwareimage = cluster.find('testimage')
```

Properties that contain lists, like `node.roles`, `node.interfaces`, `category.fsmounts` and several others, are trickier to deal with. While iterating over a list property is simple enough:

```
for role in node.roles:
  print role.name
```

because of an implementation restriction, adding a new role requires that a local copy of the roles list be made:

```
roles = node.roles
provisioningrole = pythoncm.ProvisioningRole()   # Create a new pro\
                                                 visioning role object
roles.append(provisioningrole)
node.roles = roles            # This will update the internal\
                                roles list with the local copy
```

### 1.3.3 Creating New Objects

Creating a new node can be done with:

```
node = pythoncm.Node()
```

This is valid command, but fairly useless because a node has to be a `MasterNode`, `PhysicalNode` or `VirtualSMPNode`. So to create a normal compute or login node, the object is created as follows:

```
node = pythoncm.PhysicalNode()
```

The first thing to do after creating a new object is to add it to a cluster.

```
cluster.add(node)
```

It is impossible to add one node to more than one cluster.

After the node has been added its properties can be set. In `cmsh` and `cmgui` this is semi-automated, but in Python cluster manager bindings it has to be done by hand.

```
node.hostname = 'node001'
node.partition = cluster.find('base')
node.category = cluster.find('default')
```

Similar to the node object, a `NetworkInterface` object has several subtypes: `NetworkPhysicalInterface`, `NetworkVLANInterface`, `NetworkAliasInterface`, `NetworkBondInterface`, and `NetworkIPMIInterface`.

```
interface = pythoncm.NetworkPhysicalInterface()
interface.name = 'eth0'
interface.ip = '10.141.0.1'
interface.network = cluster.find('internalnet')
node.interfaces = [interface]
node.provisioningInterface = interface
```

Having set the properties of the new node, it can now be committed.

```
cr = node.commit()
```

If a commit fails for some reason, the reason can be found:

```
if not cr.result:
  print "Commit of %s failed:" % node.resolveName()
  for j in range(cr.count):
    print cr.getValidation(j).msg
```

### 1.3.4 List Of Objects

In the following lists of objects:

- Objects marked with (*) cannot be used
- Trees marked with (+) denote inheritance

#### Roles

```
Role (*)
+ BackupRole
+ BootRole
+ DatabaseRole
+ EthernetSwitch
+ LoginRole
+ LSFClientRole
+ LSFServerRole
+ MasterRole
+ PbsProClientRole
+ PbsProServerRole
+ ProvisioningRole
+ SGEClientRole
+ SGEServerRole
+ SlurmClientRole
+ SlurmServerRole
+ SubnetManagerRole
+ TorqueClientRole
+ TorqueServerRole
```

#### Devices

```
Device (*)
+ Chassis
+ GpuUnit
+ GenericDevice
+ PowerDistributionUnit
+ Switch (*)
   + EthernetSwitch
   + IBSwitch
   + MyrinetSwitch
   Node (*)
   +FSExport
   +FSMount
   + MasterNode
   + SlaveNode (*)
     + PhysicalNode
     + VirtualSMPNode
```

#### Network Interfaces

```
NetworkInterface (*)
+ NetworkAliasInterface
+ NetworkBondInterface
+ NetworkIpmiInterface
+ NetworkPhysicalInterface
+ NetworkVLANInterface
```

#### Information Objects

```
ClusterSetup
GuiClusterOverview
GuiCephOverview
GuiHadoopHDFSOverview
GuaOpenStackOverview
GuiOpenStackTenantOverview
GuiGpuUnitOverview
GuiNodeOverview
GuiNodeStatus
LicenseInfo
SysInfoCollector
VersionInfo
```

#### LDAP Objects

```
User
Group
```

#### Category Objects

```
Category
FSExport
FSMount
```

#### Miscellaneous Objects

```
SoftwareImage

KernelModule

Network
```

```
NodeGroup

Partition
+ BurnConfig
Rack
```

### 1.3.5 Useful Methods

**For The Cluster Object:**

| Name | Description |
|---|---|
| `find(<name>)` | Find the object with a given name, <*name*> |
| `find(<name>, <type>)` | Because it is possible to give a category and node the same name, sometimes the type <*type*> of the object needs to be specified too |
| `getAll(<type>)` | Get a list of all objects of a given type: e.g. device, category |
| `activeMaster()` | Get the active master object |
| `passiveMaster()` | Get the active master object |
| `overview()` | Get all the data shown in the `cmgui` cluster overview |
| `add(<object>)` | Add a newly created object <*object*> to the cluster. Only after an object is added can it be used |
| `pexec(<nodes>, <command>)` | Execute a command <*command*> on one or more nodes |

**For Any Object:**

| Name | Description |
|---|---|
| `commit()` | Save changes to the cluster |
| `refresh()` | Undo all changes and restore the object to its last saved state |
| `remove()` | Remove an object from the cluster |
| `clone()` | Make an identical copy. The newly created object is not added to a cluster yet |

**For Any Device:**

| Name | Description |
|---|---|
| `close()` | Close a device |
| `open()` | Open a device |
| `powerOn()` | Power on a device |
| `powerOff()` | Power off a device |
| `powerReset()` | Power reset a device |
| `latestMonitoringData()` | Return a list of the most recent monitoring data |

**For Any Node:**

| Name | Description |
|---|---|
| `overview()` | Get the data displayed in the `cmgui` node overview tab |
| `sysinfo()` | Get the data displayed in the `cmgui` node system information tab |
| `pexec(<command>)` | Execute a command |

### 1.3.6 Useful Example Program

In the directory `/cm/local/examples/cmd/python` are some example programs using the python API.

One of these is `printall.py`. It displays values for objects in an easily viewed way. With `all` as the argument, it displays resource objects defined in a list in the program. The objects are 'Partition', 'MasterNode', 'SlaveNode', 'Category', 'SoftwareImage', 'Network', 'NodeGroup'. The output is displayed something like (some output elided):

**Example**

```
[root@bright81 ~]# cd /cm/local/examples/cmd/python
[root@bright81 python]# ./printall all
Partition base
 +- revision .....................
 |  name ........................ base
 |  clusterName ................. Bright 8.1 Cluster
...
 |  burnConfigs
 |   +- revision .................
 |   |  name ..................... default
 |   |  description ............... Standard burn test.
 |   |  configuration ............. < 2780 bytes >
 |   +- revision .................
 |   |  name ..................... long-hpl
...
 |  provisioningInterface ......... None
 |  fsmounts ..................... < none >
 |  fsexports
 |   +- revision .................
 |   |  name ..................... /cm/shared@internalnet
 |   |  path ..................... /cm/shared
 |   |  hosts .................... !17179869185!
...
Category default
 +- revision .....................
 |  name ........................ default
 |  softwareImage ............... default-image
 |  defaultGateway .............. 10.141.255.253
 |  nameServers ................. < none >
...
```

The values of a particular resource-level object, such as `softwareimage`, can be viewed by specifying it as the argument:

**Example**

```
[root@bright81 python]# ./printall.py softwareimage
softwareimage default-image
 +- revision ...........................
 |  name ............................... default-image
 |  path ............................... /cm/images/default-image
 |  originalImage ...................... 0
 |  kernelVersion ...................... 2.6.32-431.11.2.el6.x86_64
 |  kernelParameters ................... rdblacklist=nouveau
 |  creationTime ....................... 1398679806
 |  modules
```

```
 |   +- revision .....................
 |   |  name ........................... xen-netfront
...
 |   +- revision .................
 |   |  name ..................... hpilo
 |   |  parameters ................
 |  enableSOL ..................... False
 |  SOLPort ....................... ttyS1
 |  SOLSpeed ...................... 115200
 |  SOLFlowControl ................ True
 |  notes ........................
 |  fspart ........................ 98784247812
 |  bootfspart .................... 98784247813
...
[root@bright81 python]#
```

## 1.4 The Workload Management API

The workload management API allows the submission of jobs, the retrieval of information on jobs and queues, and the management of jobs and queues. The methods described in this section are a part of the `cmjob` service. They can also be accessed via the `Cluster` object, with exception of the `getParentJobs` and `getJobsSlice` methods.

Workload management examples for a particular workload manager <*wlm*> in Python can be found on the head node in the directory:

`/cm/local/examples/cmd/python/workload-`<*wlm*>`.py`

Here, <*wlm*> can take the values `torque`, `slurm`, `sge`, `pbspro`, `openlava`, or `lsf`. The examples define a job, with different job properties associated with different workload managers. With the right properties set, the job can be submitted and the submitted job outputs are printed to STDOUT.

Details of entities and their properties can be found in the CMDaemon API reference.

### 1.4.1 Job Submission

Job submission is performed with the `submitJob` method. Its only argument is the `Job` entity that provides the properties and resource requirements of the job that is submitted.

Each workload manager uses its own job properties format, although they usually behave in a similar way. The following table shows the correspondence between `Job` entity parameters and the submission parameters for each workload manager.

| Parameter | Slurm | PBS Pro<br>Torque | LSF<br>openlava | UGE<br>OGS (SGE) |
|---|---|---|---|---|
| `queue` | `-p` | `-q` | `-q` | `-q` |
| `jobname` | `-J` | `-N` | `-J` | `-N` |

...continues

*...continued*

| Parameter | Slurm | PBS Pro Torque | LSF openlava | UGE OGS (SGE) |
|---|---|---|---|---|
| account | -A | -A | N/A | -A |
| project | N/A | -P | -P | -P |
| rundirectory | -D | -w | N/A | -wd |
| username | Job script is submitted by this user | | | |
| groupname | Job script is submitted with group permissions of this user | | | |
| priority | --nice | -p | -sp | -p |
| stdinfile | -i | N/A | -i | -i |
| stdoutfile | -o | -o | -o | -o |
| stderrfile | -e | -e | -e | -e |
| dependencies | -d | -W depend= | -w | --hold_jid |
| mailNotify | Enables passing other email options, not used directly | | | |
| mailOptions | --mail-type | -m | -B | -m |
| mailList | --mail-user | -M | -u | -M |
| resourceList | -C | -l | -R | -l |

*...continues*

*...continued*

| Parameter | Slurm | PBS Pro Torque | LSF openlava | UGE OGS (SGE) |
|---|---|---|---|---|
| maxWallClock | -t | -l walltime= | -c | -l h_rt= |
| numberOfProcesses | -n | mpiprocs= ppn= | -n | -pe |
| numberOfNodes | -N | -l select= | -R 'span[hosts=]' | N/A |
| nodes | -w | -l select= | -m | -l hostname= |
| environmentVariables | All additional environment variables are passed to the job | | | |
| commandLineInterpreter | Interpreter path is added as a first line into the jobscript | | | |
| executable | Added as a command at the end of a new created jobscript. | | | |
| arguments | Appended to executable line | | | |
| modules | Module files will be added to job script environment | | | |
| userdefined | These lines are added into the jobscript before the executable line | | | |
| scriptFile | If scriptfile is specified, then only is it submitted | | | |
| debug | Return debug info (without submission), including generated script | | | |

Notes:

1. In the case of LSF and OpenLava, the rundirectory parameter of the Job entity is converted into a cd command line, that is added to the job script before any commands.
2. The executable file path and its arguments are translated to a single line in the job script. If more complex commands are required then the parameter userdefined should be used instead of executable and arguments. If userdefined is not an empty list, then executable and arguments are ignored.

### 1.4.2 Job Information And Management

For job manipulation the following functions are used. In these functions, the parameter <*scheduler*> is the name of the workload manager that the operation is applied to, and takes a value of `slurm`, `uge`, `sge`, `openlava`, `lsf`, `torque` or `pbspro`. The parameter <*JobID*> is a string in a format related to that particular workload manager.

**getJobs**(<*scheduler*>): returns `Job` entities for the specified scheduler. This function triggers a call to the workload manager utility. The workload manager utility is, for example, `qstat` in the case of SGE or Torque, and `scontrol` in the case of Slurm. In profiles (section 6.4 of the *Administrator Manual*), `GET_JOB_TOKEN` is needed to be able to get all the jobs, while `GET_OWN_JOB_TOKEN` is needed to get just all the jobs belonging to the user making the call.

**getJob**(<*scheduler*>, <*JobID*>): returns a job by job ID. `GET_JOB_TOKEN` is needed to be able to get any job, and `GET_OWN_JOB_TOKEN` is needed to be able to get just the job belonging to the user making the call.

**removeJob**(<*scheduler*>, <*JobID*>): removes the job by job ID and returns the result of job removal. `UPDATE_JOB_TOKEN` is needed to be able to remove any job, and `UPDATE_OWN_JOB_TOKEN` is needed to be able to remove just the job belonging to the user making the call.

**getJobsSlice**(<*scheduler*>, <*start*>, <*maxCount*>, <*parentID*>, <*allUsers*>): returns jobs at the position <*start*> in the global list (sorted by job ID), but only up to <*maxCount*> items. That is, if the value of the parameter <*start*> is a number *n*, then jobs starting from the *n*th item in the global list are returned, up to <*maxCount*> times. <*parentID*> is a method to group jobs by a keyword in the comment string of the jobs. <*allUsers*> specifies, using the value `True` or `False`, whether the jobs of all users should be considered—a value of `False` means that only the jobs owned by the requestor are considered. `GET_JOB_TOKEN` is needed to get any job slice, while `GET_OWN_JOB_TOKEN` is needed to get just the job slices belonging to the user making the call.

**getParentJobs**(<*scheduler*>, <*start*>, <*maxCount*>, <*parentID*>, <*allUsers*>): returns `parentJob` entities at the position <*start*> in the global list (sorted by parent job ID), but only up to `maxCount` items. That is, if the value of the parameter <*start*> is a number *n*, then jobs starting from the *n*th item in the global list are returned, up to <*maxCount*> times. <*parentID*> is a method to group jobs by a keyword in the comment string of the jobs. By default it has an empty value passed to it. If <*parentID*> is given a parent ID value, then the parent job is treated as owned by particular user if and only if all jobs with this tag (parent id) are submitted by that user. Setting <*allUsers*> specifies, using the value `True` or `False`, whether the jobs of all users should be considered—a value of `False` means that only the jobs owned by the requestor are considered. `GET_JOB_TOKEN` is needed to get any job slice, while `GET_OWN_JOB_TOKEN` is needed to get just the job slices belonging to the user making the call.

**requeueJob**(<*scheduler*>, <*JobID*>): requeues job and returns the result of this operation as a string. `REQUEUE_JOB_TOKEN` is needed to be able to requeue any job, while `REQUEUE_OWN_JOB_TOKEN` is needed to be able to requeue just the job belonging to the user making the call.

**holdJob**(<*scheduler*>, <*JobID*>): holds the job and returns the result of this operation as a string. `HOLD_JOB_TOKEN` is needed to be able to hold any job, while `HOLD_OWN_JOB_TOKEN` is needed to be able to hold just the job belonging to the user making the call.

**suspendJob**(<*scheduler*>, <*JobID*>): suspends the job and returns the result of this operation as a string. `SUSPEND_JOB_TOKEN` is needed to be able to suspend any job, while `SUSPEND_OWN_JOB_TOKEN` is needed to be able to suspend just the job belonging to the user making the call.

**resumeJob**(<*scheduler*>, <*JobID*>): resumes the job and returns the result of this operation as a string. `RESUME_JOB_TOKEN` is needed to be able to resume any job, while `RESUME_OWN_JOB_TOKEN` is needed to be able to resume just the job belonging to the user making the call.

**releaseJob**(<*scheduler*>, <*JobID*>): release the job and returns the result of this operation as a string. `RELEASE_JOB_TOKEN` is needed to be able to release any job, while `RELEASE_OWN_JOB_TOKEN` is needed to be able to release just the job belonging to the user making the call.

**updateJob**(<*scheduler*>, <*JobID*>): update the job and returns result of this operation as a string. `UPDATE_JOB_TOKEN` is needed to be able to update any job, while `UPDATE_OWN_JOB_TOKEN` is needed to be able to update just the job belonging to the user making the call.

**isNodeAllocatedForUser**(<*scheduler*>, <*username*>, <*hostname*>): returns true if at least one job owned by the user, as specified by the value of <*username*> allocates the host, as specified by the value of <*hostname*>.

Parent job is an entity introduced in Bright 7.3 and serves a goal of jobs clusterization. The jobs can be united by a tag surrounded by square brackets (for example "*[workflow1]*"). The tag is parsed by CMDaemon from the job comment line. The first entry of such a tag in the job comment is considered as the parent job ID. CMDaemon caches parent jobs, and an API client can request all the parent jobs or just some particular one. This allows the client to unite jobs by some user-defined property in a workflow, even if the workload manager does not support the workflow.

### 1.4.3 Queue Information And Management

For queue manipulation the following functions are used.

**getJobQueues**(): retrieves all `JobQueue` entities. Requires `GET_JOBQUEUE_TOKEN`.

**getJobQueue**(<*queuename*>): retrieves a particular `JobQueue` entity. Here <*queuename*> is a string. Requires `GET_JOBQUEUE_TOKEN`.

**getParallelEnvs**(<*scheduler*>): retrieves a list of ParallelEnvironment entities associated with a particular workload manager. Requires `GET_PE_TOKEN`.

**getJobQueueStates**(): retrieves a list of `JobQueueStat` entities. Requires `GET_JOBQUEUE_TOKEN`.

**updateJobQueue**(<*JobQueue*>, <*force*>): updates job queue properties defined by JobQueue entity. Parameter <*force*> is ignored for now. Requires `UPDATE_JOBQUEUE_TOKEN`.

**addJobQueue**(<*JobQueue*>, <*force*>): adds a new job queue to workload manager. If <*force*> has the value `True`, then the existing queue is recreated. Requires `ADD_JOBQUEUE_TOKEN`.

**removeJobQueue**(<*queueKey*>, <*force*>): removes queue by key. The key can be retrieved from the `JobQueue` entity requested by the `getJobQueue` method. Parameter <*force*> is ignored for now. Requires `UPDATE_JOBQUEUE_TOKEN`.

**drainNodes**(<*scheduler*>, <*queue*>, <*nodes*>, <*drain*>)**:** drains nodes (as defined by a list of hostnames or uniqueKeys) or a particular queue (if supported by the workload manager) in the workload manager. If <*drain*> has the value `1`, then the nodes will be drained, otherwise they are undrained. Returns a list `DrainResult` entities. Requires `DRAIN_TOKEN`.

**drainOverview**(<*scheduler*>, <*nodes*>)**:** returns `DrainResult` entities with current drain state of the nodes. The nodes are defined by a list of hostnames or uniqueKeys. Requires `DRAIN_OVERVIEW_TOKEN`.

# 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 Bright Cluster Manager `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 Bright 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. Bright 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@bright81 ~]# cat /path/to/my/metric
#!/bin/bash
echo $((RANDOM))
# Optionally provide extra information
echo "Extra information" >&3
```

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

**Example**

```
[bright81]% monitoring setup
[bright81->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@bright81 ~]# cat /path/to/my/health-check
#!/bin/bash
if [ $((RANDOM)) -gt 8000 ]; then
  echo "PASS"
else
  echo "FAIL"
  # Optionally provide extra information
  echo "Extra information" >&3
fi
```

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

**Example**

```
[bright81]% monitoring setup
[bright81->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@bright81 ~]# 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**

```
[bright81]% monitoring setup
[bright81->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@bright81 ~]# 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@bright81 ~]# 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**

```
[bright81]% monitoring setup
[bright81->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**

```
[bright81]% monitoring setup
[bright81->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**

```
[bright81]% 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**

```
[bright81]% 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**

```
[bright81]% 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**

```
[bright81]% 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 Bright Cluster Manager 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**

```
[bright81]% device use node001
[bright81]% monitoringresources
CentOS7u5
Ethernet
category:default
```

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

**Example**

```
[bright81]% device use node001
[bright81]% add userdefinedresources MyResource
[bright81]% append userdefinedresources MyOtherResource
[bright81]% # wait ~10 seconds for the settings to propagate
[bright81]% 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@bright81~]# 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**

```
[bright81]% monitoring setup
[bright81->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 Bright Cluster Manager entity.

For this reason the standalone monitored entity was added in Bright Cluster Manager 8.0.

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

Bright Cluster Manager 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**

```
[bright81]% monitoring standalone
[bright81->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@bright81 ~]# 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**

```
[bright81]% monitoring setup
[bright81->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 Bright Cluster Manager nodes.

**Example**

```
[bright81]% monitoring standalone
[bright81->monitoring->standalone]% use MSD.0
[...standalone*[MSD.0*]]% latestmetricdata
...
lustre.free.space        12345         3m 47s
```

# 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**

```
[bright81]% monitoring action
[bright81->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**

```
[bright81]% monitoring action
[bright81->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-friday9:00-17:00
monday-friday00:00-09:00;17:00-00:00;saturday-sunday
november-marchmonday-saturday13:00-17:00
may-septembermonday-friday09:00-18:00;saturday-sunday13:00-17:00
```

Further examples can be seen in section 12.4.4 of the *Administrator Manual*, page 457.

### 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_KEY | The unique key 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. |
| 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. |

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.

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 additionally available environment variables with some example values:

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

| Variable | Example Value |
|---|---|
| CMD_ACTIVE_MASTER_IP | 10.141.255.254 |

*...continues*

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

| Variable | Example Value |
|---|---|
| CMD_ADDED_NODES | |
| CMD_BASE_TYPE | |
| CMD_BMCIP | |
| CMD_BMCPASSWORD | doQNeV1qksXr59O |
| CMD_BMCUSERID | 4 |
| CMD_BMCUSERNAME | |
| CMD_BMC_TYPE | 2 |
| CMD_CATEGORY | default |
| CMD_CEPH_MDS_SOCKET | |
| CMD_CEPH_MGR_SOCKET | |
| CMD_CEPH_MON_SOCKET | |
| CMD_CEPH_NAME | |
| CMD_CEPH_OSD_ID | |
| CMD_CEPH_OSD_SOCKET | |
| CMD_CHASSIS | chassis01 |
| CMD_CHASSIS_IP | 10.141.1.1 |
| CMD_CHASSIS_MEMBERS | |
| CMD_CHASSIS_PASSWORD | secr3t |
| CMD_CHASSIS_SLOT | 1 |

...continues

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

| Variable | Example Value |
|---|---|
| CMD_CHASSIS_USERNAME | ADMIN |
| CMD_CHILD_TYPE | |
| CMD_CLUSTERNAME | Bright 8.1 Cluster |
| CMD_CONFIGURATION_CREATE_DIRECTORY | |
| CMD_CONFIGURATION_FILENAME | |
| CMD_CONFIGURATION_GROUP_NAME | |
| CMD_CONFIGURATION_MASK | |
| CMD_CONFIGURATION_NAME | |
| CMD_CONFIGURATION_USER_NAME | |
| CMD_CREATE_RAMDISK_TOKEN_CATS | |
| CMD_CREATE_RAMDISK_TOKEN_NODES | |
| CMD_CURRENT_NODES | |
| CMD_DATA | |
| CMD_DELLFW_FTP_PASSWORD | |
| CMD_DELLFW_FTP_USERNAME | |
| CMD_DELLFW_PATH | |
| CMD_DESTINATION_REVISION | |
| CMD_DESTINATION_VERSION | |
| CMD_DEVICE_HEIGHT | 1 |

...continues

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

| Variable | Example Value |
|---|---|
| CMD_DEVICE_POSITION | 10 |
| CMD_DEVICE_TYPE | ComputeNode |
| CMD_DIRECTOR | |
| CMD_DIRECTOR_IP | |
| CMD_DOCKER_ENDPOINTS | |
| CMD_EDGE_SITE | |
| CMD_ETCD_CA | |
| CMD_ETCD_CAKEY | |
| CMD_ETCD_CLIENT_CA | |
| CMD_ETCD_CLIENT_CERT | |
| CMD_ETCD_CLIENT_KEY | |
| CMD_ETCD_MEMBER_CERT | |
| CMD_ETCD_MEMBER_KEY | |
| CMD_ETHERNETSWITCH | switch01:1 |
| CMD_EXISTING_REVISION | |
| CMD_EXISTING_VERSION | |
| CMD_EXPORTS | |
| CMD_FAILONMISSINGBMC | |
| CMD_FAIL_ON_FAILED_BMCCOMMAND | YES |

...continues

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

| Variable | Example Value |
|---|---|
| CMD_FSEXPORTS | |
| CMD_FSEXPORT_<*name*>_ALLOWWRITE | |
| CMD_FSEXPORT_<*name*>_HOSTS | |
| CMD_FSEXPORT_<*name*>_PATH | |
| CMD_FSMOUNTS | |
| CMD_FSMOUNT_<*name*>_DEVICE | |

where <*name*> takes these SLASH substitutions:

| **<*name*>** | **example value** |
|---|---|
| _SLASH_cm_SLASH_shared | $localnfsserver:/cm/shared |
| _SLASH_dev_SLASH_pts | *none* |
| _SLASH_dev_SLASH_shm | *none* |
| _SLASH_home | $localnfsserver:/home |
| _SLASH_proc | *none* |
| _SLASH_sys | *none* |

CMD_FSMOUNT_<*name*>_FILESYSTEM

where <*name*> takes these SLASH substitutions:

| **<*name*>** | **example value** |
|---|---|
| _SLASH_cm_SLASH_shared | nfs |
| _SLASH_dev_SLASH_pts | devpts |
| _SLASH_dev_SLASH_shm | tmpfs |
| _SLASH_home | nfs |
| _SLASH_proc | proc |
| _SLASH_sys | sysfs |

CMD_FSMOUNT_<*name*>_MOUNTPOINT

where <*name*> takes these SLASH substitutions:

| **<*name*>** | **example value** |
|---|---|
| _SLASH_cm_SLASH_shared | /cm/shared |
| _SLASH_dev_SLASH_pts | /dev/pts |
| _SLASH_dev_SLASH_shm | /dev/shm |
| _SLASH_home | /home |
| _SLASH_proc | /proc |
| _SLASH_sys | /sys |

*...continues*

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

| Variable | Example Value |
|---|---|
| CMD_FSMOUNT_<*name*>_OPTIONS | |

where <*name*> takes these SLASH substitutions:

| *<name>* | example value |
|---|---|
| _SLASH_cm_SLASH_shared | rsize=32768,wsize=32768,hard,intr,async |
| _SLASH_dev_SLASH_pts | gid=5,mode-620 |
| _SLASH_dev_SLASH_shm | defaults |
| _SLASH_home | rsize=32768,wsize=32768,hard,intr,async |
| _SLASH_proc | defaults,nosuid |
| _SLASH_sys | /defaults |

| | |
|---|---|
| CMD_GATEWAY | 10.141.255.254 |
| CMD_GUID | |
| CMD_HAPROXY_HOST | |
| CMD_HOSTNAME | node004 |
| CMD_INITRD | |
| CMD_INITRD_KERNEL_PARAMS | |
| CMD_INITRD_KERNEL_VERSION | |
| CMD_INITRD_TMPFS_SIZE | |
| CMD_INSTALLMODE | AUTO |
| CMD_INSTANCE_ID | |
| CMD_INTERFACES | BOOTIF |
| CMD_INTERFACE_<*interface*>_BOND | |
| CMD_INTERFACE_<*interface*>_BRIDGE | |
| CMD_INTERFACE_<*interface*>_DHCP | |
| CMD_INTERFACE_<*interface*>_GATEWAY | |

...continues

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

| Variable | Example Value |
|---|---|
| CMD_INTERFACE_<*interface*>_IP | 10.141.0.5 |
| CMD_INTERFACE_<*interface*>_LANCHANNEL | |
| CMD_INTERFACE_<*interface*>_MAC | 00:00:00:00:00:00 |
| CMD_INTERFACE_<*interface*>_MODE | |
| CMD_INTERFACE_<*interface*>_MTU | 1500 |
| CMD_INTERFACE_<*interface*>_NETMASK | |
| CMD_INTERFACE_<*interface*>_REVISION | |
| CMD_INTERFACE_<*interface*>_SLAVES | |
| CMD_INTERFACE_<*interface*>_SPEED | |
| CMD_INTERFACE_<*interface*>_STARTIF | ALWAYS |
| CMD_INTERFACE_<*interface*>_TYPE | NetworkPhysicalInterface |
| CMD_INTERFACE_<*interface*>_VLANID | |

In the preceding CMD_INTERFACE_* variables, <*interface*> can take the following substitutions for the network interface:

| possible values for <*interface*> |
|---|
| BOOTIF |
| drac0, drac1, drac2... |
| cimc0, cimc1, cimc2... |
| eth0, eth1, eth1... |
| ib0, ib1, ib2... |
| ilo0, ilo1, ilo2... |
| ipmi0, ipmi1, ipmi2... |
| rf0, rf1, rf2... |
| eno1, enp0s18f2, and other names consistent with the RHEL7 interface naming convention |

| | |
|---|---|
| CMD_IP | 10.141.0.1 |
| CMD_JOBNODELIST | |
| CMD_KUBERNETES_ADMIN_CERT | |

...continues

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

| Variable | Example Value |
|---|---|
| CMD_KUBERNETES_ADMIN_CERT_KEY | |
| CMD_KUBERNETES_ADMIN_KUBECONFIG | |
| CMD_KUBERNETES_APISERVER_ENDPOINT | |
| CMD_KUBERNETES_CACERT | |
| CMD_KUBERNETES_CLIENT_CERTIFICATE | |
| CMD_KUBERNETES_CLIENT_KEY | |
| CMD_KUBERNETES_ETCD_ACTIVE | |
| CMD_KUBERNETES_ETCD_CLIENT_ENDPOINTS | |
| CMD_KUBERNETES_KUBELET_CERTIFICATE | |
| CMD_KUBERNETES_KUBELET_ENDPOINT | |
| CMD_KUBERNETES_KUBELET_KEY | |
| CMD_KUBE_DNS_IP | |
| CMD_KUBE_DOMAIN | |
| CMD_KUBE_INTERNAL_NETWORK_CIDR | |
| CMD_KUBE_POD_NETWORK_CIDR | |
| CMD_KUBE_SERVICE_NETWORK_CIDR | |
| CMD_LOGGING_CONFIG | |
| CMD_MAC | FA:16:3E:64:8E:1E |
| CMD_MODEL | |

...continues

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

| Variable | Example Value |
|---|---|
| CMD_MODULES | |
| CMD_MODULE_<*name*> | |
| CMD_MOUNTS | |
| CMD_NAME | |
| CMD_NODEGROUPS | |
| CMD_NODEGROUP_NAME | |
| CMD_NODEGROUP_UID | |
| CMD_OPENSTACK_CINDER_PASSWORD | |
| CMD_OPENSTACK_CINDER_USERNAME | |
| CMD_OPENSTACK_ENABLED | |
| CMD_OPENSTACK_GLANCE_PASSWORD | |
| CMD_OPENSTACK_GLANCE_USERNAME | |
| CMD_OPENSTACK_KEYSTONE_INTERNAL_URL | |
| CMD_OPENSTACK_KEYSTONE_PASSWORD | |
| CMD_OPENSTACK_KEYSTONE_PUBLIC_URL | |
| CMD_OPENSTACK_KEYSTONE_URL | |
| CMD_OPENSTACK_KEYSTONE_USERNAME | |
| CMD_OPENSTACK_MANAGER_PASSWORD | |
| CMD_OPENSTACK_MANAGER_USERNAME | |

*...continues*

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

| Variable | Example Value |
|---|---|
| CMD_OPENSTACK_MESSAGE_QUEUE_HOSTS | |
| CMD_OPENSTACK_MESSAGE_QUEUE_PASSWORD | |
| CMD_OPENSTACK_MESSAGE_QUEUE_USERNAME | |
| CMD_OPENSTACK_NOVA_PASSWORD | |
| CMD_OPENSTACK_NOVA_USERNAME | |
| CMD_OPENSTACK_REGIONS | |
| CMD_OPENSTACK_SERVICE_TENANT | |
| CMD_OPENSTACK_SSL_ENABLED | |
| CMD_OPENSTACK_TENANTS | |
| CMD_OWNED_INDEX | |
| CMD_PARTITION | base |
| CMD_PASSIVE_MASTER_IP | 10.141.255.253 |
| CMD_PDUS | |
| CMD_PORT | 8081 |
| CMD_PORTS | |
| CMD_POWER_CONTROL | custom |
| CMD_PROTOCOL | https |
| CMD_RACADM_PATH | |
| CMD_RACK | rack01 |

*...continues*

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

| Variable | Example Value |
|---|---|
| CMD_RACK_HEIGHT | 42 |
| CMD_RACK_ROOM | serverroom |
| CMD_READ_STRING | |
| CMD_REMOVED_NODES | |
| CMD_RESOLVE_NAME | |
| CMD_ROLES | |
| CMD_SCRIPTTIMEOUT | 5 |
| CMD_SCRIPT_TIMEOUT | 5 |
| CMD_SHARED_MASTER_IP | 10.141.255.252 |
| CMD_SKIPBMC | |
| CMD_SOFTWAREIMAGE | default-image |
| CMD_SOFTWAREIMAGE_PATH | /cm/images/default-image |
| CMD_STATE | |
| CMD_STATUS | |
| CMD_STATUS_CLOSED | NO |
| CMD_STATUS_HEALTHCHECK_FAILED | NO |
| CMD_STATUS_HEALTHCHECK_UNKNOWN | NO |
| CMD_STATUS_MESSAGE | |
| CMD_STATUS_RESTART_REQUIRED | NO |

...continues

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

| Variable | Example Value |
|---|---|
| CMD_STATUS_STATEFLAPPING | NO |
| CMD_STATUS_USERMESSAGE | |
| CMD_STRICTUSERID | |
| CMD_SUBNET_MANAGER | |
| CMD_SWITCH_CONTROL_SCRIPT | |
| CMD_SWITCH_CONTROL_SCRIPT_TIMEOUT | |
| CMD_SYSINFO_SYSTEM_MANUFACTURER | RDO |
| CMD_SYSINFO_SYSTEM_NAME | OpenStack Compute |
| CMD_TAG | 00000000a000 |
| CMD_TARGET_NAME | |
| CMD_TARGET_NODES | |
| CMD_TYPE | |
| CMD_TYPES | |
| CMD_UCS_DN | sys/rack-unit-1 |
| CMD_USERDEFINED1 | var1 |
| CMD_USERDEFINED2 | var2 |
| CMD_VMLINUZ | |
| CMD_WRITE_STRING | |

# 4

# Bright Cluster Manager JSON API

This chapter gives an alphabetical list of the JSON API services and entities available for Bright Cluster Manager. The API reference documentation for all available services and entities is available on the head node at:

`/cm/local/apps/cmd/etc/htdocs/userportal/download/json/index.html`.

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 12.7.3 of the *Administrator Manual*).

Some examples of JSON use are given in section 4.3

## 4.1 Services

## 4.2 Entities

## 4.3 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-cert\
ificate --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='uniqueKey'

node=master
```

```
json=`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.uniqueKey = $nkey"

json=`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.uniqueKey = $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" : $((now*1000))
              }
}
EOF
wget --certificate=$HOME/.cm/admin.pem --private-key=$HOME/.cm/admin.key \
--no-check-certificate -qO- https://master:8081/json --post-file=/tmp/plot.json | \
python -mjson.tool
```

**loadone.sh**

```
#!/bin/bash
source url

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

prop='uniqueKey'

node=master
```

```
json='wget --load-cookies cookie.txt --no-check-certificate --se\
rver-response -qO- $URL --post-data='{"service":"cmdevice","call\
":"getDevice","arg1":"'$node'"}'`
nkey=$(jsonval)
if [ -z $nkey ]; then
  echo $json
  exit 1
fi
echo "$node.uniqueKey = $nkey"

json='wget --load-cookies cookie.txt --no-check-certificate --se\
rver-response -qO- $URL --post-data='{"service":"cmmon","call":"\
getMetric","arg1":"loadOne"}'`
mkey=$(jsonval)
echo "loadone.uniqueKey = $mkey"

now=$(date +%s)
day=$((now-86400))


# echo ------------------------------------------------------------
# wget --load-cookies cookie.txt --no-check-certificate --server\
-response -qO- $URL \
#   --post-data='{"service":"cmmon","call":"readDataByIntervalNu\
m",
#                 "readMonDataIdArray":[{"devId":'$nkey',"metric\
Id":'$mkey',
#                                        "begTime":'$day',"endTi\
me":'$now'}],
#                 "intervalNum":0}'
#
# echo
echo ------------------------------------------------------------
wget --load-cookies cookie.txt --no-check-certificate --server-r\
esponse -qO- $URL \
  --post-data='{"service":"cmmon","call":"readDataByIntervalNum",
               "args":[[{"baseType":"ReadMonDataId","uniqueKey"\
:0,"modified":false,"toBeRemoved":false,"childType":"",
                        "devId":'$nkey',"metricId":'$mkey',
                        "begTime":'$day',"endTime":'$now'}],0]}'
# echo
# echo ------------------------------------------------------------
# data='{"service":"cmmon","call":"readDataByIntervalNum",
#                 "args":[[{"baseType":"ReadMonDataId","uniqueKe\
y":0,"modified":false,"toBeRemoved":false,"childType":"",
#                           "devId":'$nkey',"metricId":'$mkey',
#                           "begTime":'$day',"endTime":'$now'}],\
0]}'
# rm loadone.txt.gz
# echo $data > loadone.txt
# gzip -n loadone.txt
# len=$(wc -c loadone.txt.gz | cut -d" " -f1)
# wget --load-cookies cookie.txt --no-check-certificate --header\
 "Content-Length: $len" --header 'Content-Encoding: gzip' --serv\
er-response -O- $URL \
```

```
#   --post-file=loadone.txt.gz
```

**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
source url

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"}'
```