Cisco VIM REST API

The following topics explain how to use the Cisco VIM REST API to manage Cisco NFVI.

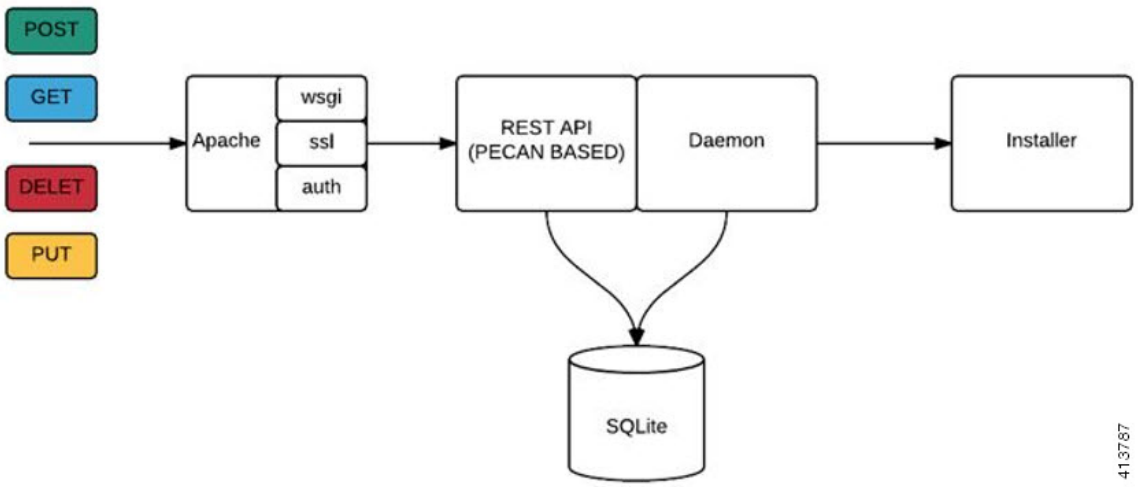
Overview to Cisco VIM REST API

Cisco VIM provides a Representational State Transfer (REST) API that is used to install, expand, and update Cisco VIM. Actions performed using the REST APIs are:

- Install Cisco VIM on Cisco NFVI pods
- Add and delete pods to and from Cisco NFVI installations
- Update Cisco VIM software
- Replace controller nodes
- Perform cloud maintenance operations
- Run cloud validations using Virtual Machine ThroughPut (VMTP), a data path performance measurement tool for OpenStack clouds

The following figure shows the workflow of Cisco VIM REST API.

Figure 1: Workflow of Cisco VIM REST API



The Cisco VIM REST API security is provided by the Secure Sockets Layer (SSL) included on the Apache web server. The Pecan-based web application is called by mod_wsgi, which runs the Rest API server. The Pecan REST API server requires a username and password to authorize the REST API server requests. Apache handles the authorization process, which authorizes the request to access the Pecan web application. Use the Cisco VIM API to upload a new setup_data.yaml file, and start, stop, and query the state of the installation. You can use it to manage the cloud, add and remove compute and Ceph nodes, and replace the controller nodes. A REST API to launch VMTP (L2/L3 data plane testing) and CloudPulse is also provided.

The Cisco VIM REST API is enabled by default in the management node if you are using the supplied Cisco VIM buildnode.iso. You can access API server on the br_api interface on port 8445. Authentication is enabled by default in the web service.

You can access the API end points using the following URL format:

```
https://<Management_node_api_ip>:8445
```

By default, basic authentication is enabled for the API endpoints in the management node. You can find the authentication credentials in the following file in the management node:

```
/opt/cisco/ui_config.json
```

The following code shows a `sample ui_config.json` file.

```
{
  "Kibana-Url": "http://10.10.10.10:5601",
  "RestAPI-Url": "https:// 10.10.10.10:8445",
  "RestAPI-Username": "admin",
  "RestAPI-Password": "a96e86ccb28d92ceb1df",
  "RestDB-Password": "e32de2263336446e0f57",
  "BuildNodeIP": "10.10.10.10"
}
```

For more information on the Rest API for an end-point, see the *Ciscovim Client RestAPI* section in Troubleshooting.

Cisco VIM REST API Resources

Setupdata

REST wrapper for setupdata. Provides methods for listing, creating, modifying, and deleting setupdata.

Retrieving the setupdata

Resource URI

Verb	URI
GET	/v1/setupdata

Example

JSON Request

```
GET /v1/setupdata
Accept: application/json
```

JSON Response

```
200 OK
Content-Type: application/json
{"setupdatas": [{
     "status": "Active",
     "name":"GG34",
     "uuid": "123"
     "meta":{
           "user":"root"
     },
     "jsondata":{
       .......
     }
 }]}
```

Creating the setupdata

Resource URI

Verb	URI
POST	/v1/setupdata

Example

JSON Request

```
POST /v1/setupdata
Accept: application/json

{
    "name":"GG34",
    "uuid": "123"
    "meta":{
         "user":"root"
  },
  "jsondata":{
    .......
```

```
  }
}
```

JSON Response

```
201 OK
Content-Type: application/json
{
     "status": "Active",
     "name":"GG34",
     "uuid": "123"
     "meta":{
         "user":"root"
  },
  "jsondata":{
    .......
  }
}

400 Bad Request
Content-Type: application/json
{
    "debuginfo": null
    "faultcode": "Client"
    "faultstring": "Error"
}

409 CONFLICT
Content-Type: application/json
{
     "debuginfo": null
     "faultcode": "Client"
     "faultstring": "Error"
}
```

Retrieving a single setupdata

Resource URI

Verb	URI
GET	/v1/setupdata/(id)

Property:

id—The ID of the setupdata that you want to retrieve.

Example

JSON Request

```
GET /v1/setupdata/123
Accept: application/json
```

JSON Response

```
200 OK
Content-Type: application/json
{
     "status": "Active",
     "name":"GG34",
     "uuid": "123"
```

```
     "meta":{
         "user":"root"
  },
  "jsondata":{
    .......
  }
}

404 NOT FOUND
Content-Type: application/json
{
     "debuginfo": null
     "faultcode": "Client"
     "faultstring": "Setupdata could not be found."
}
```

Updating a setupdata

Resource URI

Verb	URI
PUT	/v1/setupdata/(id)

Property:

id—The ID of the setupdata that you want to update.

Example

JSON Request

```
PUT /v1/setupdata/123
Accept: application/json
```

JSON Response

```
200 OK
Content-Type: application/json
{
     "status": "Active",
     "name":"GG34",
     "uuid": "123"
     "meta":{
         "user":"root"
  },
  "jsondata":{
    .......
  }
}

404 NOT FOUND
Content-Type: application/json
{
     "debuginfo": null
     "faultcode": "Client"
     "faultstring": "Setupdata could not be found."
}
```

Deleting a setupdata

Resource URI

Verb	URI
DELETE	/v1/setupdata/(id)

Property:

id—The ID of the setupdata that you want to delete.

Example

JSON Request

```
DELETE /v1/setupdata/123
Accept: application/json
```

JSON Response

```
204 NO CONTENT
Returned on success

404 NOT FOUND
Content-Type: application/json
{
    "debuginfo": null
    "faultcode": "Client"
    "faultstring": "Setupdata could not be found."
}
400 BAD REQUEST
Content-Type: application/json

{
     "debuginfo": null
     "faultcode": "Client"
     "faultstring": "Setupdata cannot be deleted when it is being used by an installation"
}
```

Install resource

REST wrapper for install. Provides methods for starting, stopping, and viewing the status of the installation process.

Return a list of installation

Resource URI

Verb	URI
GET	/v1/install

Example

JSON Request

```
GET /v1/install
Accept: application/json
```

JSON Response

```
200 OK
Content-Type: application/json
```

```
{"installs": [{
      "ceph": "Skipped",
      "uuid": "123",
      "setupdata": "345",
      "vmtpresult": "{
         "status": "PASS",
         "EXT_NET": []
      }",
      "baremetal": "Success",
      "orchestration": "Success",
      "validationstatus": "{
         "status": "PASS",
         "Software_Validation": [],
         "Hardware_Validation": []
      }",
      "currentstatus": "Completed",
      "validation": "Success",
      "hostsetup": "Success",
      "vmtp": "Skipped"
    }]
}
```

Create an installation

Resource URI

Verb	URI
POST	/v1/install

Example

JSON Request

```
GET /v1/install
Accept: application/js
{
     "setupdata": "123",
     "stages": [
       "validation",
       "bootstrap",
       "runtimevalidation",
       "baremetal",
       "orchestration",
       "hostsetup",
       "ceph",
       "vmtp"
     ]
}
```

JSON Response

```
201  CREATED
Content-Type: application/json
{
    "ceph": "Skipped",
    "uuid": "123",
    "setupdata": "345",
    "vmtpresult": "{
       "status": "PASS",
       "EXT_NET": []
    }",
```

```
    "baremetal": "Success",
    "orchestration": "Success",
    "validationstatus": "{
       "status": "PASS",
       "Software_Validation": [],
       "Hardware_Validation": []
    }",
     "currentstatus": "Completed",
     "validation": "Success",
     "hostsetup": "Success",
     "vmtp": "Skipped"
 }


409 CONFLICT
Content-Type: application/json
{
     "debuginfo": null
     "faultcode": "Client"
     "faultstring": "Install already exists"
}
```

Retrieve the installation

Resource URI

Verb	URI
GET	/v1/install/{id}

Property:

id—The ID of the installation that you want to retrieve.

Example

JSON Request

```
GET /v1/install/345
Accept: application/js
```

JSON Response

```
200  OK
Content-Type: application/json
{
    "ceph": "Skipped",
    "uuid": "123",
    "setupdata": "345",
    "vmtpresult": "{
       "status": "PASS",
       "EXT_NET": []
    }",
    "baremetal": "Success",
    "orchestration": "Success",
    "validationstatus": "{
      "status": "PASS",
      "Software_Validation": [],
      "Hardware_Validation": []
    }",
    "currentstatus": "Completed",
    "validation": "Success",
```

```
    "hostsetup": "Success",
    "vmtp": "Skipped"
}


404 NOT FOUND
Content-Type: application/json
{
     "debuginfo": null
     "faultcode": "Client"
     "faultstring": "Install doesn't exists"
}
```

Stop the installation

Resource URI

Verb	URI
DELETE	/v1/install/{id}

Property:

id—The ID of the installation that you want to stop.

Example

JSON Request

```
DELETE /v1/install/345
Accept: application/js
```

JSON Response

```
204 NO CONTENT
Content-Type: application/json

404 NOT FOUND
Content-Type: application/json
{
     "debuginfo": null
     "faultcode": "Client"
     "faultstring": "Install doesn't exists"
}
```

Nodes

Getting a list of nodes

Resource URI

Verb	URI
GET	/v1/nodes

Example

JSON Request

```
Get /v1/nodes
Accept: application/js
```

JSON Response

```
200 OK
Content-Type: application/json
{
    "nodes": [
        [
           "status": "Active",
           "uuid": "456",
           "setupdata": "123",
           "node_data": "{
             "rack_info": {
                "rack_id": "RackA"
             },
             "cimc_info": {
               "cimc_ip": "10.10.10.10"
             },
             "management_ip": "7.7.7.10"
             }",
             "updated_at": null,
             "mtype": "compute",
             "install": "345",
             "install_logs": "logurl",
             "created_at":"2016-0710T06:17:03.761152",
             "name": " compute-1"
             }
          ]
}
```

Add New Nodes

The nodes are in compute or block_storage type. Before adding the nodes to the system, the name of the nodes and other necessary information like cimc_ip and rackid must be updated in the setupdata object. If the setupdata object is not updated, the post call does not allow you to add the node.

Resource URI

Verb	URI
POST	/v1/nodes

Example

JSON Request

```
POST /v1/nodes
Accept: application/js
{
    "name" : "compute-5"
}
```

JSON Response

```
201 CREATED
Content-Type: application/json
{
     "status": "ToAdd",
     "uuid": "456",
     "setupdata": "123",
```

```
     "node_data": "{
        "rack_info": {
        "rack_id": "RackA"
        },
        "cimc_info": {
         "cimc_ip": "10.10.10.10"
        },
        "management_ip": "7.7.7.10"
        }",
        "updated_at": null,
        "mtype": "compute",
        "install": "345",
        "install_logs": "logurl",
        "created_at":"2016-0710T06:17:03.761152",
        "name": " compute-1"
}
```

Retrieve information about a particular node

Resource URI

Verb	URI
GET	/v1/nodes{id}

Property:

id—The ID of the node that you want to retrieve.

Example

JSON Request

```
POST /v1/nodes
Accept: application/js
```

JSON Response

```
200 OK
Content-Type: application/json
{
     "status": "Active",
     "uuid": "456",
     "setupdata": "123",
     "node_data": "{
       "rack_info": {
        "rack_id": "RackA"
       },
       "cimc_info": {
         "cimc_ip": "10.10.10.10"
       },
       "management_ip": "7.7.7.10"
       }",
       "updated_at": null,
       "mtype": "compute",
       "install": "345",
       "install_logs": "logurl",
       "created_at":"2016-0710T06:17:03.761152",
       "name": " compute-1"
}

404 NOT FOUND
```

```
Content-Type: application/json
{
    "debuginfo": null
    "faultcode": "Client"
    "faultstring": "Node doesn't exists"
}
```

Remove a Node

The node that must be deleted must be removed from the setupdata object. Once the setupdata object is updated, you can safely delete the node. The node object cannot be deleted until it calls the remove node backend and succeeds.

Resource URI

Verb	URI
DELETE	/v1/nodes{id}

Property:

id—The ID of the node that you want to remove.

Example

JSON Request

```
DELETE /v1/nodes/456
Accept: application/js
```

JSON Response

```
204 ACCEPTED
Content-Type: application/json

404 NOT FOUND
Content-Type: application/json
{
     "debuginfo": null
     "faultcode": "Client"
     "faultstring": "Node doesn't exists"
}
```

To clear the database and delete the entries in the nodes, the delete API is called with special parameters that are passed along with the delete request. The JSON parameters are in the following format.

JSON Request

```
DELETE /v1/nodes/456
Accept: application/js
{
    "clear_db_entry":"True"\
}
```

JSON Response

```
204 ACCEPTED
Content-Type: application/json
```

```
404 NOT FOUND
Content-Type: application/json
{
    "debuginfo": null
    "faultcode": "Client"
    "faultstring": "Node doesn’t exists"
}
```

Note This is done only if the node is deleted from the REST API database. The failure reason of the node must be rectified manually apart from the API. True is a string and not a boolean in the preceding line.

Replace a controller

Resource URI

Verb	URI
PUT	/v1/nodes{id}

Property:

id—The ID of the controller that you want to replace.

Example

JSON Request

```
PUT /v1/nodes/456
Accept: application/js
```

JSON Response

```
200 OK
Content-Type: application/json

404 NOT FOUND
Content-Type: application/json
{
    "debuginfo": null
    "faultcode": "Client"
    "faultstring": "Node doesn’t exists"
}
```

Offline validation

REST wrapper does the offline validation of setupdata. Rest Wrapper does only the Software Validation of the input setupdata.

Create an offline validation operation

Resource URI

Verb	URI
POST	/v1/offlinevalidation

Example

JSON Request

```
POST /v1/offlinevalidation
Accept: application/json
{
      "jsondata": ".. .. .."
}
```

JSON Response

```
201 CREATED
Content-Type: application/json
{
     "status": "NotValidated",
     "uuid": "bb42e4ba-c8b7-4a5c-98b3-1f384aae2b69",
     "created_at": "2016-02-03T02:05:28.384274",
     "updated_at": "2016-02-03T02:05:51.880785",
     "jsondata": "{}",
     "validationstatus": {
        "status": "PASS",
        "Software_Validation": [],
        "Hardware_Validation": []
      }
}
```

Retrieve the results of offline validation

Resource URI

Verb	URI
GET	/v1/offlinevalidation

Property:

id—The ID of the node you want to retrieve.

Example

JSON Request

```
GET /v1/offlinevalidation/789
Accept: application/json
```

JSON Response

```
200 OK
Content-Type: application/json
{
    "status": " ValidationSuccess",
    "uuid": "bb42e4ba-c8b7-4a5c-98b3-1f384aae2b69",
    "created_at": "2016-02-03T02:05:28.384274",
    "updated_at": "2016-02-03T02:05:51.880785",
    "jsondata": "{}",
    "validationstatus": {
       "status": "PASS",
       "Software_Validation": [],
       "Hardware_Validation": []
     }
```

```
}
```

Update

Start an Update Process

Resource URI

Verb	URI
POST	/v1/update

Parameters:

- fileupload - "tar file to upload"
- filename - "Filename being uploaded"

Example

JSON Request

```
curl -sS -X POST --form
"fileupload=@Test/installer.good.tgz" --form
"filename=installer.good.tgz"
https://10.10.10.8445/v1/update
```

Note This curl request is done as a form request.

JSON Response

```
200 OK
Content-Type: application/json
{
    "update_logs": "logurl",
    "update_status": "UpdateSuccess",
    "update_filename": "installer-4579.tgz",
    "created_at": "2016-07-10T18:33:52.698656",
    "updated_at": "2016-07-10T18:54:56.885083"
}

409 CONFLICT
Content-Type: application/json
{
    "debuginfo": null
    "faultcode": "Client"
    "faultstring": "Uploaded file is not in tar format"
 }
```

Roll back an update

Resource URI

Verb	URI
PUT	/v1/update

Example

JSON Request

```
PUT /v1/update
Accept: application/json
{
     "action":"rollback"
}
```

JSON Response

```
200 OK
Content-Type: application/json
{
    "update_logs": "logurl",
    "update_status": "ToRollback",
    "update_filename": "installer-4579.tgz",
    "created_at": "2016-07-10T18:33:52.698656",
    "updated_at": "2016-07-10T18:54:56.885083"
}
```

Commit an update

Resource URI

Verb	URI
PUT	/v1/update

Example

JSON Request

```
PUT /v1/update
Accept: application/json
{
"action":"commit"
}
```

JSON Response

```
200 OK
Content-Type: application/json
{
    "update_logs": "logurl",
    "update_status": "ToCommit",
    "update_filename": "installer-4579.tgz",
    "created_at": "2016-07-10T18:33:52.698656",
    "updated_at": "2016-07-10T18:54:56.885083"
}
```

Retrieve the details of an update

Resource URI

Verb	URI
GET	/v1/update

Example

JSON Request

```
GET /v1/update
Accept: application/json
```

JSON Response

```
200 OK
Content-Type: application/json
{
    "update_logs": "logurl",
    "update_status": "UpdateSuccess",
    "update_filename": "installer-4579.tgz",
    "created_at": "2016-07-10T18:33:52.698656",
    "updated_at": "2016-07-10T18:54:56.885083"
}
```

Secrets

Retrieve the list of secrets that are associated with the OpenStack Setup

You can retrieve the set of secret password that are associated with the OpenStack setup using the preceding api. This gives the list of secrets for each service in OpenStack.

Resource URI

Verb	URI
GET	/v1/secrets

Example

JSON Request

```
GET /v1/secrets
Accept: application/json
```

JSON Response

```
200 OK
Content-Type: application/json
{
"HEAT_KEYSTONE_PASSWORD": "xxxx",
"CINDER_KEYSTONE_PASSWORD": "xxxxx",
….
….
"RABBITMQ_PASSWORD": "xxxxx"
}
```

OpenStack Configs

Retrieve the list of OpenStack configs associated with the OpenStack Setup

You can retrieve the set of OpenStack configs associated with the OpenStack setup using the preceding api. This gives the current settings of different configs such as verbose logging, debug logging for different OpenStack services.

Resource URI

Verb	URI

GET	/v1/openstack_config

Example

JSON Request

```
GET /v1/openstack_config
Accept: application/json
```

JSON Response

```
200 OK
Content-Type: application/json
{
"CINDER_DEBUG_LOGGING": false,
"KEYSTONE_DEBUG_LOGGING": false,
….
….
"NOVA_VERBOSE_LOGGING": true
}
```

Version

Retrieve the version of the Cisco Virtualized Infrastructure Manager.

Resource URI

Verb	URI
GET	/v1/version

Example

JSON Request

```
GET /v1/version
Accept: application/json
```

JSON Response

```
200 OK
Content-Type: application/json
{"version": "1.9.1"}
```

Health of the Management Node

Retrieve the health of the Management node

This API is used to retrieve the health of the management node. It checks various parameters such as partitions, space and so on.

Resource URI

Verb	URI
GET	/v1/health

Example

JSON Request

```
GET /v1/health
Accept: application/json
```

JSON Response

```
200 OK
Content-Type: application/json
{
    "status": "PASS",
    "BuildNode Validation": {
        "Check Docker Pool Settings": {"status": "Pass", "reason": "None"}
        ….
        ….
    }
}
```

Hardware Information

REST wrapper to do hardware information of setupdata. This returns the hardware information of all hardware available in the setupdata.

Create a HWinfo operation

Resource URI

Verb	URI
GET	/v1/hwinfo

Example

JSON Request

```
POST /v1/hwinfo
Accept: application/json
{
       "setupdata":"c94d7973-2fcc-4cd1-832d-453d66e6b3bf"
}
```

JSON Response

```
201 CREATED
Content-Type: application/json
{
  "status": "hwinfoscheduled",
  "uuid": "928216dd-9828-407b-9739-8a7162bd0676",
  "setupdata": "c94d7973-2fcc-4cd1-832d-453d66e6b3bf",
  "created_at": "2017-03-19T13:41:25.488524",
  "updated_at": null,
  "hwinforesult": ""
}
```

Retrieve the results of Hwinfo Operation

Resource URI

Verb	URI
GET	/v1/hwinfo/{id}

Property:

id—The ID of the node you want to query.

Example

JSON Request

```
GET /v1/hwinfo/789
Accept: application/json
```

JSON Response

```
200 OK
Content-Type: application/json
{
  "status": "hwinfosuccess",
  "uuid": "928216dd-9828-407b-9739-8a7162bd0676",
  "setupdata": "c94d7973-2fcc-4cd1-832d-453d66e6b3bf",
  "created_at": "2017-03-19T13:41:25.488524",
  "updated_at": "2017-03-19T13:42:05.087491",
  "hwinforesult": "{\"172.29.172.73\": {\"firmware\": ……..
  ……
  ……..
}
```

Release mapping Information

This api is used to see the list of Features included and list of options which can be reconfigured in the Openstack Setup.

Retrieve the release mapping information

Resource URI

Verb	URI
GET	/v1/releasemapping

JSON Request

```
GET /v1/releasemapping
Accept: application/json
```

JSON Response

```
200 OK
Content-Type: application/json
[
  {
    "SWIFTSTACK": {
      "feature_status": true,
      ],
      "desc": "swift stack feature"
    }
  },……..
  ……..
}
```

POST Install operations

The following are the post install operations that can be carried on after the OpenStack installation is carried out successfully. It uses a common api. Following is an Example:

1. reconfigure,
2. reconfigure -regenerate passwords
3. reconfigure -setpasswords,setopenstack_configs
4. check-fernet-keys
5. period-rotate-fernet-keys

6. resync-fernet-keys
7. rotate-fernet-keys

Create a post install operation

Resource URI

Verb	URI
POST	/v1/misc

Example

JSON Request

```
POST /v1/misc
Accept: application/json
{"action": {"reconfigure": true}}
```

JSON Response

```
201 CREATED
Content-Type: application/json
{
  "uuid": "7e30a671-bacf-4e3b-9a8f-5a1fd8a46733",
  "created_at": "2017-03-19T14:03:39.723914",
  "updated_at": null,
  "operation_status": "OperationScheduled",
  "operation_logs": "",
  "operation_name": "{"reconfigure": true}"
}
```

Retrieve a status of the post install operation

Resource URI

Verb	URI
GET	/v1/misc

Example

JSON Request

```
GET /v1/misc
Accept: application/json
```

JSON Response

```
201 CREATED
Content-Type: application/json
{
  "uuid": "7e30a671-bacf-4e3b-9a8f-5a1fd8a46733",
  "created_at": "2017-03-19T14:03:39.723914",
  "updated_at": "2017-03-19T14:03:42.181180",
  "operation_status": "OperationRunning",
  "operation_logs": "xxxxxxxxxxxxxxxx",
  "operation_name": "{\"reconfigure\": true}"
}
```

In VIM Rest APIs exist to support NFVBench, query hardware information and to get a list of optional and mandatory features that the pod supports.

Following are the API details:

NFVBench Network Performance Testing

Create NFVBench Run

Starts the network performance test with provided configuration.

REST API To Create Fixed Rate Test

Verb	URI
Post	v1/nfvbench/ create_ndr_pdr_test

Example

JSON Request

```
POST Request URL
/v1/nfvbench/create_fixed_rate_test
JSON Request:
{“nfvbench_request”:
{
    "duration_sec": 20,
    "traffic_profile":  [
         {
            "name": "custom",
            "l2frame_size": [
               "64",
               "IMIX",
               "1518"
            ]
         }
   ],
   "traffic": {
      "bidirectional": true,
      "profile": "custom"
   },
   "flow_count": 1000
}
}
```

JSON Response

```
201 CREATED
Content-Type: application/json
 {
      "status": "not_run",
"nfvbench_request":
‘{
    "duration_sec": 20,
    "traffic_profile":  [
         {
            "name": "custom",
            "l2frame_size": [
               "64",
               "IMIX",
               "1518"
            ]
         }
   ],
   "traffic": {
      "bidirectional": true,
```

```
      "profile": "custom"
   },
   "flow_count": 1000
}',
"created_at": "2017-08-16T06:14:54.219106",
"updated_at": null,
"nfvbench_result": "",
"test_name": "Fixed_Rate_Test"
}
```

Status Polling

Polling of NFVbench run status which is one of nfvbench_running, nfvbench_failed, nfvbench_completed.

Resource URI

Verb	URI
GET	v1/nfvbench/<test_name>

REST API To Get Fixed Rate Test Result

```
GET Request URL
/v1/upgrade/get_fixed_rate_test_result
JSON Response:
 Check If NFVbench Test is running
  200 OK
  Content-Type: application/json
{
     "status": "nfvbench_running",
     "nfvbench_request": '{"traffic": {"bidirectional": true, "profile": "custom"},
"rate": "1000000pps",
"traffic_profile": [{"l2frame_size": ["1518"], "name": "custom"}], "duration_sec": 60,
"flow_count": 1000}',
"nfvbench_result": ""
     "created_at": "2017-05-30T21:40:40.394274",
        "updated_at": "2017-05-30T21:40:41.367279",
}

Check If NFVbench Test is completed
  200 OK
  Content-Type: application/json
   {
"status": "nfvbench_completed",
"nfvbench_request": '{"traffic": {"bidirectional": true, "profile": "custom"},
 "rate": "1000000pps",
"traffic_profile": [{"l2frame_size": ["1518"], "name": "custom"}], "duration_sec": 60,
"flow_count": 1000}',
"nfvbench_result": '{"status": "PROCESSED", "message": {"date": "2017-08-15 23:15:04",
"nfvbench_version": "0.9.3.dev2", ….}
"created_at": "2017-05-30T21:40:40.394274",
"updated_at": "2017-05-30T22:29:56.970779",
   }
```

REST API to create NDR/PDR Test

```
POST Request URL
/v1/nfvbench/create_ndr_pdr_test

Accept: application/json
{"nfvbench_request":
{
    "duration_sec": 20,
    "traffic_profile":  [
         {
```

```
            "name": "custom",
            "l2frame_size": [
               "64",
               "IMIX",
               "1518"
            ]
         }
   ],
   "traffic": {
      "bidirectional": true,
      "profile": "custom"
   },
   "flow_count": 1000
}
}

JSON Response
201 CREATED
Content-Type: application/json
 {
    "status": "not_run",
"nfvbench_request":
'{
    "duration_sec": 20,
    "traffic_profile":  [
         {
            "name": "custom",
            "l2frame_size": [
               "64",
               "IMIX",
               "1518"
            ]
         }
   ],
   "traffic": {
      "bidirectional": true,
      "profile": "custom"
   },
   "flow_count": 1000
}'

"created_at": "2017-08-16T07:18:41.652891",
"updated_at": null,
     "nfvbench_result": "",
     "test_name": "NDR_PDR_Test"
}
```

REST API To Get NDR/PDR Test Results

```
GET Request URL
/v1/ nfvbench/get_ndr_pdr_test_result

JSON Response:
 If NFVbench NDR/PDR test is running
    200 OK
    Content-Type: application/json
{
     "status": "nfvbench_running",
 "nfvbench_request": '{"duration_sec": 20,
 "traffic": {"bidirectional": true, "profile": "custom"},
 "traffic_profile": [{"l2frame_size": ["64", "IMIX", "1518"], "name": "custom"}],
"flow_count": 1000}',
 "nfvbench_result": ""
"created_at": "2017-08-16T07:18:41.652891",
"updated_at": "2017-09-30T22:29:56.970779",
```

```
}

If NFVbench NDR/PDR test is completed
  200 OK
  Content-Type: application/json
{
 "status": "nfvbench_completed",
 "nfvbench_request": '{"duration_sec": 20,
"traffic": {"bidirectional": true, "profile": "custom"},
"traffic_profile": [{"l2frame_size": ["64", "IMIX", "1518"], "name": "custom"}], "flow_count":
 1000}',
     "nfvbench_result": '{"status": "PROCESSED",...}'
"created_at": "2017-08-16T07:18:41.652891",
"updated_at": "2017-09-30T22:29:56.970779",

}
```

REST API to Get Node Hardware Information

Rest API helps you to get the hardware information of all the nodes in the POD through CIMC/UCSM.

- Total Memory
- Firmware Info (Model, Serial Number)
- CIMC IP

```
GET Request URL
/v1/hwinfo
Output Response
{
    "hwinforesult": "{"control-server-2": {"memory": {"total_memory": "131072"},
    "firmware": {"serial_number": "FCH1905V16Q", "fw_model": "UCSC-C220-M4S"},
    "cimc_ip": "172.31.230.100", "storage": {"num_storage": 4},
    "cisco_vic_adapters": {"product_name": "UCS VIC 1225"},
     "cpu": {"number_of_cores": "24"}, "power_supply": {"power_state": "on"}}
    …
 }
```

REST API to Get Mandatory Features Mapping

```
POST Request URL
/v1/releasemapping/mandatory_features_mapping

JSON Response:
{
    "mandatory": {
        "networkType": {
            "C": {
                "feature_status": true,
               "values": [{"name": "VXLAN/Linux Bridge", "value": "VXLAN/Linux Bridge"},],

                "insight_label": "Tenant Network",
                "desc": "Tenant Network"
            },
            "B": {
                "feature_status": true,
               "values": [{"name": "VXLAN/Linux Bridge", "value": "VXLAN/Linux Bridge"},],

                "insight_label": "Tenant Network",
                "desc": "Tenant Network"
            }
        },
        "cephMode": {
```

```
            "all": {
                "feature_status": true,
                "values": [{"name": "Central", "value": "Central"},],
                "insight_label": "Ceph Mode",
                "desc": "Ceph Mode"
            }
        },
        "podType": {
            "C": {
                "feature_status": true,
                "values": [{"name": "Fullon", "value": "fullon"},],
                "insight_label": "POD Type",
                "desc": "POD Type"
            },
            "B": {
                "feature_status": true,
                "values": [{"name": "Fullon", "value": "fullon"},],
                "insight_label": "POD Type",
                "desc": "POD Type"
            }
        },
        "installMode": {
            "all": {
                "feature_status": true,
                "values": [{"name": "Connected", "value": "connected"}, ],
                "insight_label": "Install Mode",
                "desc": "Install Mode"
            }
        }
    },
    "platformType": [{"name": "B-series", "value": "B"}, {"name": "C-series", "value":
"C"}],
    "postinstalllinks": {
       "view_cloudpulse": {"alwayson": true, "feature_status": true, "platformtype": "all",
 "insight_label": "Run VMTP", "desc": "Cloudpluse"},
        "password_reconfigure": {"alwayson": true, "feature_status": true, "platformtype":
 "all", "insight_label": "Reconfigure Passwords", "desc": "Reconfigure Passwords"}
    }
}
```


REST API to Get Optional Features Mapping

```
POST Request URL
/v1/releasemapping/optional_features_mapping

JSON Response:
 [
    {
        "SWIFTSTACK": {
            "feature_status": true,
            "insight_label": "Swiftstack",
            "repeated_redeployment": true,
           "reconfigurable": ["cluster_api_endpoint", "reseller_prefix", "admin_password",
 "protocol"],
            "desc": "swift stack feature"
        }
    },
    {
        "heat": {
            "feature_status": true,
            "insight_label": "Heat",
            "repeated_redeployment": false,
            "reconfigurable": ["all"],
            "desc": "Openstack HEAT service"
        }
```

```
    },
….. other features
]
```

Cloud sanity information

REST wrapper to run cloud-sanity test suites. The cloud-sanity extension to the VIM REST API enables support for managing cloud-sanity test actions

Create a cloud-sanity test

Verb	URI
Post	/v1/cloud-sanity/create

Example

JSON Request

```
POST /v1/cloudsanity/create
Accept: application/json
'{"cloudsanity_request": {"command": "create",
                          "action": "test",
                          "test_name": "cephmon",
                          "uuid": ""}}'

test_name can be all,management,control,compute,cephmon,cephosd
```

JSON Response

```
201 Created
{
 'cloudsanity_request': "{u'action': u'test', u'command': u'create', u'uuid':
'5dff1662-3d33-4901-808d-479927c01dde',
 u'test_name': u'cephmon'}",
'cloudsanity_result': '',
'created_at': '2018-01-26T20:32:20.436445',
'status': 'not_run',
'test_name': 'cephmon',
'updated_at': ''
}
```

List cloud-sanity test results

Verb	URI
GET	/v1/cloud-sanity

JSON Request

```
GET /v1/cloudsanity
```

JSON Response

```
200 OK
{ '0b91746f-90b4-4355-a748-727c2e5c59c5': { 'action': 'test',
                                            'created_at': '2018-01-25 12:08:22',
                                            'status': 'cloudsanity_completed',
                                            'test_name': 'management',
                                          'uuid': '0b91746f-90b4-4355-a748-727c2e5c59c5'},

  '5695cb31-39e4-4be2-9dee-09e7daffc2e7': { 'action': 'test',
                                            'created_at': '2018-01-25 12:03:06',
                                            'status': 'cloudsanity_completed',
```

```
                                          'test_name': 'compute',
                                         'uuid': '5695cb31-39e4-4be2-9dee-09e7daffc2e7'},

  '5dff1662-3d33-4901-808d-479927c01dde': { 'action': 'test',
                                          'created_at': '2018-01-26 20:32:20',
                                          'status': 'cloudsanity_completed',
                                          'test_name': 'cephmon',
                                         'uuid': '5dff1662-3d33-4901-808d-479927c01dde'},

  '7946255d-df58-4432-b729-20cf16eb5ba5': { 'action': 'test',
                                          'created_at': '2018-01-25 12:05:56',
                                          'status': 'cloudsanity_completed',
                                          'test_name': 'cephosd',
                                         'uuid': '7946255d-df58-4432-b729-20cf16eb5ba5'},

  '797d79ba-9ee0-4e11-9d9e-47791dd05e07': { 'action': 'test',
                                          'created_at': '2018-01-25 12:05:11',
                                          'status': 'cloudsanity_completed',
                                          'test_name': 'cephmon',
                                         'uuid': '797d79ba-9ee0-4e11-9d9e-47791dd05e07'},

  '962e2c8e-c7b0-4e24-87c1-528cad84002c': { 'action': 'test',
                                          'created_at': '2018-01-26 18:52:31',
                                          'status': 'cloudsanity_completed',
                                          'test_name': 'control',
                                         'uuid': '962e2c8e-c7b0-4e24-87c1-528cad84002c'},

  'd0111530-ee3b-45df-994c-a0917fd18e11': { 'action': 'test',
                                          'created_at': '2018-01-26 18:46:23',
                                          'status': 'cloudsanity_completed',
                                          'test_name': 'control',
                                         'uuid': 'd0111530-ee3b-45df-994c-a0917fd18e11'}}
```

List specific cloud-sanity test results

Verb	URI
GET	/v1/cloud-sanity/list/?test_name={all,management, control,compute,cephmon,cephosd}

JSON Request

```
GET /v1/cloudsanity/list/?test_name=cephmon
Accept: application/json
```

JSON Response

```
200 OK
{ '5dff1662-3d33-4901-808d-479927c01dde': { 'action': 'test',
                                          'created_at': '2018-01-26 20:32:20',
                                          'status': 'cloudsanity_completed',
                                          'test_name': 'cephmon',
                                         'uuid': '5dff1662-3d33-4901-808d-479927c01dde'},

  '797d79ba-9ee0-4e11-9d9e-47791dd05e07': { 'action': 'test',
                                          'created_at': '2018-01-25 12:05:11',
                                          'status': 'cloudsanity_completed',
                                          'test_name': 'cephmon',
                                         'uuid': '797d79ba-9ee0-4e11-9d9e-47791dd05e07'}}
```

Show cloud-sanity test results

Verb	URI
GET	/v1/cloud-sanity/show/?uuid=<uuid>

JSON Request

```
GET /v1/cloudsanity/show/?uuid=d0111530-ee3b-45df-994c-a0917fd18e11
```

JSON Response

```
200 OK
{ 'action': 'test',
  'cloudsanity_request':
      "{u'action': u'test',
        u'command': u'create',
        u'uuid': 'd0111530-ee3b-45df-994c-a0917fd18e11',
        u'test_name': u'control'}",
  'cloudsanity_result':
     '{"status": "PROCESSED",
       "message": {"status": "Pass",
                   "message": "[PASSED] Cloud Sanity Control Checks Passed",
                   "results": {"control": {"ping_all_controller_nodes": "PASSED",
                                           "check_rabbitmq_is_running": "PASSED",
                                           "check_rabbitmq_cluster_status": "PASSED",
                                           "check_nova_service_list": "PASSED",
                                           "ping_internal_vip": "PASSED",
                                           "disk_maintenance_raid_health": "PASSED",
                                           "check_mariadb_cluster_size": "PASSED",
                                           "disk_maintenance_vd_health": "PASSED"}}}}',
  'created_at': '2018-01-26 18:46:23',
  'status': 'cloudsanity_completed',
  'test_name': 'control',
  'updated_at': '2018-01-26 18:47:58',
  'uuid': 'd0111530-ee3b-45df-994c-a0917fd18e11'}
```

Delete cloud-sanity test results

Verb	URI
DELETE	/v1/cloud-sanity/delete/?uuid=<uuid>

JSON Request

```
GET /v1/cloudsanity/delete/?uuid=444aa4c8-d2ba-4379-b035-0f47c686d1c4
```

JSON Response

```
200 OK
{
    "status": "deleted",
    "message": "UUID 444aa4c8-d2ba-4379-b035-0f47c686d1c4 deleted from database",
    "uuid": "444aa4c8-d2ba-4379-b035-0f47c686d1c4",
    "error": "None"
}
```

Disk Maintenance information

REST wrapper to query information about RAID disks on Pod nodes. This returns the RAID disk information of all or a selection of RAID disks available in the Pod.

The disk management extension to the VIM REST API enables support for Disk Management actions

Create a Check disk operation

Resource URI

Verb	URI
POST	/v1/diskmgmt/check_disks

Example

JSON Request

```
POST /v1/diskmgmt/check_disks Accept: application/json
'{"diskmgmt_request": {"command": "create",
                       "action": "check-disks",
                       "role": "control",
                       "locator": "False",
                       "json_display": "False",
                       "servers": "", "uuid": ""}}'
```

JSON Response

```
201 Created
Content-Type: application/json
{
    'action': 'check-disks',
    'created_at': '2018-03-08T02:03:18.170849+00:00',
    'diskmgmt_request': "{u'uuid': '0729bdea-cc19-440f-8339-ab21e76be84b',
                         u'json_display': u'False',
     u'servers': u'',
     u'locator': u'False',
     u'role': u'control',
     u'action': u'check-disks',
     u'command': u'create'}",
    'diskmgmt_result': '',
    'status': 'not_run',
    'updated_at': 'None'
}
```

Create a replace disk operation

Verb	URI
POST	/v1/diskmgmt/replace_disks

Example

JSON Request

```
POST /v1/diskmgmt/replace_disks
Accept: application/json
'{"diskmgmt_request": {"command": "create",
                       "action": "replace-disks",
                       "role": "control",
                       "locator": "False",
                       "json_display": "False",
                       "servers": "", "uuid": ""}}'
```

JSON Response

```
201 Created
Content-Type: application/json
{
  "status": "not_run",
```

```
  "diskmgmt_request": "{u'uuid': 'cb353f41-6d25-4190-9386-330e971603c9',
                        u'json_display': u'False',
                        u'servers': u'',
                        u'locator': u'False',
                        u'role': u'control',
                        u'action': u'replace-disks',
                        u'command': u'create'}",
"created_at": "2018-03-09T12:43:41.289531+00:00",
"updated_at": "",
"diskmgmt_result": "",
"action": "replace-disks"}
```

List check disk operation

Verb	URI
GET	/v1/diskmgmt/list/?action= {check-disks,replace-disks \&role={all,management,control,compute}

Example

JSON Request

```
GET /v1/diskmgmt/list/?action=check-disks\&role=all
```

JSON Response

```
200 OK
Content-Type: application/json
{
    '0be7a55a-37fe-43a1-a975-cbf93ac78893': {   'action': 'check-disks',
                                                'created_at': '2018-03-05 14:45:45+00:00',
                                                'role': 'compute',
                                                'status': 'diskmgmt_completed',
                                                'uuid':
'0be7a55a-37fe-43a1-a975-cbf93ac78893'},
    '861d4d73-ffee-40bf-9348-13afc697ee3d': {   'action': 'check-disks',
                                                'created_at': '2018-03-05 14:44:47+00:00',
                                                'role': 'control',
                                                'status': 'diskmgmt_completed',
                                                'uuid':
'861d4d73-ffee-40bf-9348-13afc697ee3d'},
    'cdfd18c1-6346-47a2-b0f5-661305b5d160': {   'action': 'check-disks',
                                                'created_at': '2018-03-05 14:43:50+00:00',
                                                'role': 'all',
                                                'status': 'diskmgmt_completed',
                                                'uuid':
'cdfd18c1-6346-47a2-b0f5-661305b5d160'}
}

}
```

Show a completed diskmgmt operation

Verb	URI
GET	v1/diskmgmt/show/?uuid=<uuid>

Example

JSON Request

```
GET /v1/diskmgmt/show/?uuid=d24036c6-4557-4c12-8695-a92f6f9315ed
```

JSON Response

```
200 OK
Content-Type: application/json
{   'action': 'check-disks',
    'created_at': '2018-03-07 21:46:41+00:00',
    'diskmgmt_request': "{u'uuid': 'd24036c6-4557-4c12-8695-a92f6f9315ed',
                         u'json_display': False,
     u'servers': u'f24-michigan-micro-2',
     u'locator': False,
     u'role': u'compute',
     u'action': u'check-disks',
     u'command': u'create'}",
   'diskmgmt_result': '{"status": "PROCESSED", "message": ["{\'Overall_Status\': \'PASS\',
 \'Result\': {\'fcfg_disks_results_list\': [], \'spare_disks_results_list\': [],
\'raid_results_list\': [{\'RAID level\': \'RAID1\', \'Disk Med\': \'HDD\', \'server\':
\'7.7.7.6\', \'RAID type\': \'HW\', \'host\': \'f24-michigan-micro-2\', \'role\':
\'block_storage control compute\', \'VD health\': \'Optl\', \'Num VDs\': 1, \'Num PDs\':
8, \'RAID health\': \'Opt\'}], \'bad_disks_results_list\': [], \'rbld_disks_results_list\':
 [], \'add_as_spares_disks_results_list\': []}}"]}',
    'role': 'compute',
    'status': 'diskmgmt_completed',
    'updated_at': '2018-03-07 21:47:35+00:00',
    'uuid': 'd24036c6-4557-4c12-8695-a92f6f9315ed'
}
```

Delete a completed diskmgmt operation

Verb	URI
DELETE	v1/diskmgmt/delete/?uuid=<uuid>

Example

JSON Request

```
DELETE /v1/diskmgmt/delete/?uuid=d24036c6-4557-4c12-8695-a92f6f9315ed
```

JSON Response

```
200 OK
Content-Type: application/json
{
 "status": "deleted",
 "message": "UUID d24036c6-4557-4c12-8695-a92f6f9315ed deleted from database",
 "uuid": "d24036c6-4557-4c12-8695-a92f6f9315ed",
 "error": "None"

}
```

OSD Maintenance information

REST wrapper to query information about OSD on Pod storage nodes. This returns to the OSD status information of all or a selection of OSDs available in the Pod.

Create a OSD disk operation

Verb	URI
POST	/v1/osdmgmt/check_osds

Example

JSON Request

```
POST /v1/osdmgmt/osdmgmt/check_osds
'{"osdmgmt_request": {"command": "create",
                      "action": "check-osds",
                      "locator": "False",
                      "json_display": "False",
                      "servers": "",
                      "osd": "None",
                      "uuid": ""}}'
```

JSON Response

```
201 Created
Content-Type: application/json
{
    'action': 'check-osds',
    'created_at': '2018-03-08T21:26:15.329195+00:00',
    'osdmgmt_request': "{u'uuid': '9c64ee52-bed5-4b69-91a2-d589411dd223', u'json_display':
 u'False', u'servers': u'', u'locator': u'False', u'command': u'create', u'action':
u'check-osds', u'osd': u'None'}",
    'osdmgmt_result': '',
    'status': 'not_run',
    'updated_at': 'None'
}
```

Create a replace OSD operation

Verb	URI
POST	v1/osdmgmt/replace_osd

Example

JSON Request

```
POST /v1/osdmgmt/replace_osd
Accept: application/json
'{"osdmgmt_request": {"command": "create",
                      "action": "replace-osd",
                      "locator": "False",
                      "json_display": "False",
                      "servers": "f24-michigan-micro-1",
                      "osd": "osd.9",
                      "uuid": ""}}'
```

JSON Response

```
201 Created
Content-Type: application/json
{
  "status": "not_run",
  "osdmgmt_request": "{u'uuid': '5140f6fb-dca3-4801-8c44-89b293405310', u'json_display':
u'False', u'servers': u'f24-michigan-micro-1', u'locator': u'False', u'command': u'create',
 u'action': u'replace-osd', u'osd': u'osd.9'}",
  "created_at": "2018-03-09T15:07:10.731220+00:00",
  "updated_at": null,
  "action": "replace-osd",
  "osdmgmt_result": ""
}

}
```

List check OSD operation

Verb	URI
GET	v1/osdmgmt/list/?action= {check-osds,replace-osd}

Example

JSON Request

```
GET /v1/osdmgmt/list/?action=check-osds
```

JSON Response

```
200 OK
Content-Type: application/json
{
    '4efd0be8-a76c-4bc3-89ce-142de458d844': {   'action': 'check-osds',
                                                'created_at': '2018-03-08 21:31:01+00:00',
                                                'status': 'osdmgmt_running',
                                                'uuid':
'4efd0be8-a76c-4bc3-89ce-142de458d844'},
    '5fd4f9b5-786a-4a21-a70f-bffac70a3f3f': {   'action': 'check-osds',
                                                'created_at': '2018-03-08 21:11:13+00:00',
                                                'status': 'osdmgmt_completed',
                                                'uuid':
'5fd4f9b5-786a-4a21-a70f-bffac70a3f3f'},
    '9c64ee52-bed5-4b69-91a2-d589411dd223': {   'action': 'check-osds',
                                                'created_at': '2018-03-08 21:26:15+00:00',
                                                'status': 'osdmgmt_completed',
                                                'uuid':
'9c64ee52-bed5-4b69-91a2-d589411dd223'}
}


}
```

Show a completed osdmgmt operation

Verb	URI
GET	v1/osdmgmt/show/?uuid=<uuid>

Example

JSON Request

```
GET /v1/osdmgmt/show/?uuid=9c64ee52-bed5-4b69-91a2-d589411dd223
```

JSON Response

```
200 OK
Content-Type: application/json
{
    'action': 'check-osds',
    'created_at': '2018-03-08 21:26:15+00:00',
    'osdmgmt_request': "{u'uuid': '9c64ee52-bed5-4b69-91a2-d589411dd223', u'json_display':
 u'False', u'servers': u'', u'locator': u'False', u'command': u'create', u'action':
u'check-osds', u'osd': u'None'}",
    'osdmgmt_result': '{"status": "PROCESSED", "message": ["{\'Overall_Status\': \'PASS\',
 \'Result\': {  ommitted for doc }}]}',
    'status': 'osdmgmt_completed',
    'updated_at': '2018-03-08 21:27:16+00:00',
    'uuid': '9c64ee52-bed5-4b69-91a2-d589411dd223'
```

```
}

}
```

Delete a completed osdmgmt operation

Verb	URI
DELETE	v1/osdmgmt/delete/?uuid=<uuid>

Example

JSON Request

```
DELETE /v1/osdmgmt/delete/?uuid=9c64ee52-bed5-4b69-91a2-d589411dd223
```

JSON Response

```
200 OK
Content-Type: application/json
{
    'error': 'None',
    'message': 'UUID 9c64ee52-bed5-4b69-91a2-d589411dd223 deleted from database',
    'status': 'deleted',
    'uuid': '9c64ee52-bed5-4b69-91a2-d589411dd223'
}

}
```

Hardware Management Utility

REST wrapper to control the execution of or query information from the hardware validation utility.

Create a Validate Operation

Verb	URI
POST	/v1/hardwaremgmt/validate

JSON Request

```
POST /v1/hardwaremgmt/validate
'{"hwmgmt_request": {"command": "create",
                     "action": "validate",
                     "hosts": "None",
                     "file": "None",
                     "feature_list": "all",
                     "uuid": ""}}'

feature_list is a comma separated list of valid features for the given POD
```

JSON Reponse

```
201 Created
Content-Type: application/json
{
    'action': 'validate',
    'created_at': '2018-03-08T22:01:22.195232+00:00',
    'hwmgmt_request': "{u'feature_list': u'all', u'command': u'create', u'file': None,
u'action': u'validate', u'hosts': None, u'uuid': '89e094d8-b246-4620-afca-ba3529385cac'}",
    'hwmgmt_result': '',
    'status': 'not_run',
    'updated_at': 'None'
}
```

Create a Validate Operation for Failure

Verb	URI
GET	/v1/hardwaremgmt/resolve_failures

JSON Request

```
POST /v1/hardwaremgmt/resolve_failures
 {
    "hwmgmt_request": {
        "command": "create",
        "action": "resolve-failures",
        "hosts": "None",
        "file": "None",
        "feature_list": "all",
        "uuid": ""}
}
feature_list is a comma separated list of valid features for the given POD
```

JSON Response

```
201 Created
Content-Type: application/json
{
  "status": "not_run",
  "created_at": "2018-03-09T15:47:36.503712+00:00",
  "hwmgmt_request": "{u'feature_list': u'all', u'command': u'create', u'file': None,
u'action': u'resolve-failures', u'hosts': None, u'uuid':
'49dc1dc9-3170-4f68-b152-0f99bd19f7b1'}",
  "updated_at": "",
  "action": "resolve-failures",
  "hwmgmt_result": ""
 }
```

Create a Validate Operation

Verb	URI
GET	v1/hardwaremgmt/list

JSON Request

```
GET /v1/hardwaremgmt/list
```

JSON Response

```
200 OK
Content-Type: application/json
{   '89e094d8-b246-4620-afca-ba3529385cac': {   'action': 'validate',
                                                'created_at': '2018-03-08 22:01:22+00:00',
                                                'feature_list': 'all',
                                                'status': 'hardwaremgmt_completed',
                                                'uuid':
'89e094d8-b246-4620-afca-ba3529385cac'},
    '9f70e872-a888-439a-8661-2d2f36a4f4b1': {   'action': 'validate',
                                                'created_at': '2018-03-08 20:34:32+00:00',
                                                'feature_list': 'all',
                                                'status': 'hardwaremgmt_completed',
                                                'uuid':
'9f70e872-a888-439a-8661-2d2f36a4f4b1'}
}
```

Show a completed hardwaremgmt operation

Verb	URI
GET	/v1/hardwaremgmt/show /?uuid=<uuid>

JSON Request

```
GET /v1/hardwaremgmt/show/?uuid=9f70e872-a888-439a-8661-2d2f36a4f4b
```

JSON Response

```
200 OK
Content-Type: application/json
{
    'action': 'validate',
    'created_at': '2018-03-08 20:34:32+00:00',
    'feature_list': 'all',
    'hwmgmt_request': "{u'feature_list': u'all', u'hosts': None, u'file': None, u'action':
 u'validate', u'command': u'create', u'uuid': '9f70e872-a888-439a-8661-2d2f36a4f4b1'}",
    'hwmgmt_result': '{"status": "PROCESSED", "message": "Validate of all completed",
"results": {"status": "PASS", "results": [{"status": "PASS", "name": "CIMC Firmware Version
 Check", "err": null}, {"status": "PASS", "name": "All Onboard LOM Ports Check", "err":
null}, {"status": "PASS", "name": "PCIe Slot: HBA Status Check", "err": null}, {"status":
"PASS", "name": "Server Power Status Check", "err": null}, {"status": "PASS", "name": "NFV
 Config Check", "err": null}, {"status": "PASS", "name": "Physical Drives Check", "err":
null}, {"status": "PASS", "name": "PCIe Slot(s) OptionROM Check", "err": null}, {"status":
 "PASS", "name": "Intel Network Adapter Check", "err": null}]}}',
    'status': 'hardwaremgmt_completed',
    'updated_at': '2018-03-08 20:38:02+00:00',
    'uuid': '9f70e872-a888-439a-8661-2d2f36a4f4b1'
```

Delete a completed hardwaremgmt operation

Verb	URI
DELETE	/v1/hardwaremgmt/delete/?uuid=<uuid>

JSON Request

```
DELETE /v1/hardwaremgmt/delete/?uuid=9f70e872-a888-439a-8661-2d2f36a4f4b1
```

JSON Response

```
200 OK
Content-Type: application/json
{
    'error': 'None',
    'message': 'UUID 9f70e872-a888-439a-8661-2d2f36a4f4b1 deleted from database',
    'status': 'deleted',
    'uuid': '9f70e872-a888-439a-8661-2d2f36a4f4b1'
}
```