ODL OVSDB

Release master

Jun 20, 2019

Contents

This documentation provides critical information needed to help you to use OVSDB as a southbound plugin.

Contents:

CHAPTER 1

OVSDB User Guide

The OVSDB project implements the OVSDB protocol (RFC 7047), as well as plugins to support OVSDB Schemas, such as the Open_vSwitch database schema and the hardware_vtep database schema.

1.1 OVSDB Plugins

1.1.1 Overview and Architecture

There are currently two OVSDB Southbound plugins:

- odl-ovsdb-southbound: Implements the OVSDB Open_vSwitch database schema.
- odl-ovsdb-hwvtepsouthbound: Implements the OVSDB hardware_vtep database schema.

These plugins are normally installed and used automatically by higher level applications such as odl-ovsdb-openstack; however, they can also be installed separately and used via their REST APIs as is described in the following sections.

1.1.2 OVSDB Southbound Plugin

The OVSDB Southbound Plugin provides support for managing OVS hosts via an OVSDB model in the MD-SAL which maps to important tables and attributes present in the Open_vSwitch schema. The OVSDB Southbound Plugin is able to connect actively or passively to OVS hosts and operate as the OVSDB manager of the OVS host. Using the OVSDB protocol it is able to manage the OVS database (OVSDB) on the OVS host as defined by the Open_vSwitch schema.

OVSDB YANG Model

The OVSDB Southbound Plugin provides a YANG model which is based on the abstract network topology model.

The details of the OVSDB YANG model are defined in the ovsdb.yang file.

The OVSDB YANG model defines three augmentations:

ovsdb-node-augmentation This augments the network-topology node and maps primarily to the Open_vSwitch table of the OVSDB schema. The ovsdb-node-augmentation is a representation of the OVS host. It contains the following attributes.

- **connection-info** - holds the local and remote IP address and TCP port numbers for the OpenDaylight to OVSDB node connections
- **db-version** - version of the OVSDB database
- **ovs-version** - version of OVS
- **list managed-node-entry** - a list of references to ovsdb-bridge-augmentation nodes, which are the OVS bridges managed by this OVSDB node
- **list datapath-type-entry** - a list of the datapath types supported by the OVSDB node (e.g. *system*, *netdev*) - depends on newer OVS versions
- **list interface-type-entry** - a list of the interface types supported by the OVSDB node (e.g. *internal*, *vxlan*, *gre*, *dpdk*, etc.) - depends on newer OVS verions
- **list openvswitch-external-ids** - a list of the key/value pairs in the Open_vSwitch table external_ids column
- **list openvswitch-other-config** - a list of the key/value pairs in the Open_vSwitch table other_config column
- **list managery-entry** - list of manager information entries and connection status
- **list qos-entries** - list of QoS entries present in the QoS table
- **list queues** - list of queue entries present in the queue table

ovsdb-bridge-augmentation This augments the network-topology node and maps to an specific bridge in the OVSDB bridge table of the associated OVSDB node. It contains the following attributes.

- **bridge-uuid** - UUID of the OVSDB bridge
- **bridge-name** - name of the OVSDB bridge
- **bridge-openflow-node-ref** - a reference (instance-identifier) of the OpenFlow node associated with this bridge
- **list protocol-entry** - the version of OpenFlow protocol to use with the OpenFlow controller
- **list controller-entry** - a list of controller-uuid and is-connected status of the OpenFlow controllers associated with this bridge
- **datapath-id** - the datapath ID associated with this bridge on the OVSDB node
- **datapath-type** - the datapath type of this bridge
- **fail-mode** - the OVSDB fail mode setting of this bridge
- **flow-node** - a reference to the flow node corresponding to this bridge
- **managed-by** - a reference to the ovsdb-node-augmentation (OVSDB node) that is managing this bridge
- **list bridge-external-ids** - a list of the key/value pairs in the bridge table external_ids column for this bridge
- **list bridge-other-configs** - a list of the key/value pairs in the bridge table other_config column for this bridge

ovsdb-termination-point-augmentation This augments the topology termination point model. The OVSDB Southbound Plugin uses this model to represent both the OVSDB port and OVSDB interface for a given port/interface in the OVSDB schema. It contains the following attributes.

- **port-uuid** - UUID of an OVSDB port row

- **interface-uuid** - UUID of an OVSDB interface row
- **name** - name of the port and interface
- **interface-type** - the interface type
- **list options** - a list of port options
- **ofport** - the OpenFlow port number of the interface
- **ofport_request** - the requested OpenFlow port number for the interface
- **vlan-tag** - the VLAN tag value
- **list trunks** - list of VLAN tag values for trunk mode
- **vlan-mode** - the VLAN mode (e.g. access, native-tagged, native-untagged, trunk)
- **list port-external-ids** - a list of the key/value pairs in the port table external_ids column for this port
- **list interface-external-ids** - a list of the key/value pairs in the interface table external_ids interface for this interface
- **list port-other-configs** - a list of the key/value pairs in the port table other_config column for this port
- **list interface-other-configs** - a list of the key/value pairs in the interface table other_config column for this interface
- **list inteface-lldp** - LLDP Auto Attach configuration for the interface
- **qos** - UUID of the QoS entry in the QoS table assigned to this port

Getting Started

To install the OVSDB Southbound Plugin, use the following command at the Karaf console:

```
feature:install odl-ovsdb-southbound-impl-ui
```

After installing the OVSDB Southbound Plugin, and before any OVSDB topology nodes have been created, the OVSDB topology will appear as follows in the configuration and operational MD-SAL.

HTTP GET:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/
 or
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/
```

Result Body:

```
{
  "topology": [
    {
      "topology-id": "ovsdb:1"
    }
  ]
}
```

Where

<controller-ip> is the IP address of the OpenDaylight controller

OpenDaylight as the OVSDB Manager

An OVS host is a system which is running the OVS software and is capable of being managed by an OVSDB manager. The OVSDB Southbound Plugin is capable of connecting to an OVS host and operating as an OVSDB manager. Depending on the configuration of the OVS host, the connection of OpenDaylight to the OVS host will be active or passive.

Active Connection to OVS Hosts

An active connection is when the OVSDB Southbound Plugin initiates the connection to an OVS host. This happens when the OVS host is configured to listen for the connection (i.e. the OVSDB Southbound Plugin is active the the OVS host is passive). The OVS host is configured with the following command:

```
sudo ovs-vsctl set-manager ptcp:6640
```

This configures the OVS host to listen on TCP port 6640.

The OVSDB Southbound Plugin can be configured via the configuration MD-SAL to actively connect to an OVS host.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1
```

Body:

```
{
  "network-topology:node": [
    {
      "node-id": "ovsdb://HOST1",
      "connection-info": {
        "ovsdb:remote-port": "6640",
        "ovsdb:remote-ip": "<ovs-host-ip>"
      }
    }
  ]
}
```

Where

<ovs-host-ip> is the IP address of the OVS Host

Note that the configuration assigns a *node-id* of "ovsdb://HOST1" to the OVSDB node. This *node-id* will be used as the identifier for this OVSDB node in the MD-SAL.

Query the configuration MD-SAL for the OVSDB topology.

HTTP GET:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/
```

Result Body:

```
{
  "topology": [
    {
      "topology-id": "ovsdb:1",
```

(continues on next page)

(continued from previous page)

```
      "node": [
        {
          "node-id": "ovsdb://HOST1",
          "ovsdb:connection-info": {
            "remote-ip": "<ovs-host-ip>",
            "remote-port": 6640
          }
        }
      ]
    }
  ]
}
```

As a result of the OVSDB node configuration being added to the configuration MD-SAL, the OVSDB Southbound Plugin will attempt to connect with the specified OVS host. If the connection is successful, the plugin will connect to the OVS host as an OVSDB manager, query the schemas and databases supported by the OVS host, and register to monitor changes made to the OVSDB tables on the OVS host. It will also set an external id key and value in the external-ids column of the Open_vSwtich table of the OVS host which identifies the MD-SAL instance identifier of the OVSDB node. This ensures that the OVSDB node will use the same *node-id* in both the configuration and operational MD-SAL.

```
"opendaylight-iid" = "instance identifier of OVSDB node in the MD-SAL"
```

When the OVS host sends the OVSDB Southbound Plugin the first update message after the monitoring has been established, the plugin will populate the operational MD-SAL with the information it receives from the OVS host.

Query the operational MD-SAL for the OVSDB topology.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/
```

Result Body:

```
{
  "topology": [
    {
      "topology-id": "ovsdb:1",
      "node": [
        {
          "node-id": "ovsdb://HOST1",
          "ovsdb:openvswitch-external-ids": [
            {
              "external-id-key": "opendaylight-iid",
              "external-id-value": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb://HOST1']"
            }
          ],
          "ovsdb:connection-info": {
            "local-ip": "<controller-ip>",
            "remote-port": 6640,
            "remote-ip": "<ovs-host-ip>",
            "local-port": 39042
          },
          "ovsdb:ovs-version": "2.3.1-git4750c96",
```

(continues on next page)

(continued from previous page)

```
          "ovsdb:manager-entry": [
            {
              "target": "ptcp:6640",
              "connected": true,
              "number_of_connections": 1
            }
          ]
        }
      ]
    }
  ]
}
```

To disconnect an active connection, just delete the configuration MD-SAL entry.

HTTP DELETE:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1
```

Note in the above example, that / characters which are part of the *node-id* are specified in hexadecimal format as "%2F".

Passive Connection to OVS Hosts

A passive connection is when the OVS host initiates the connection to the OVSDB Southbound Plugin. This happens when the OVS host is configured to connect to the OVSDB Southbound Plugin. The OVS host is configured with the following command:

```
sudo ovs-vsctl set-manager tcp:<controller-ip>:6640
```

The OVSDB Southbound Plugin is configured to listen for OVSDB connections on TCP port 6640. This value can be changed by editing the "./karaf/target/assembly/etc/custom.properties" file and changing the value of the "ovsdb.listenPort" attribute.

When a passive connection is made, the OVSDB node will appear first in the operational MD-SAL. If the Open_vSwitch table does not contain an external-ids value of *opendaylight-iid*, then the *node-id* of the new OVSDB node will be created in the format:

```
"ovsdb://uuid/<actual UUID value>"
```

If there an *opendaylight-iid* value was already present in the external-ids column, then the instance identifier defined there will be used to create the *node-id* instead.

Query the operational MD-SAL for an OVSDB node after a passive connection.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/
```

Result Body:

```
{
  "topology": [
    {
```

(continues on next page)

(continued from previous page)

```
      "topology-id": "ovsdb:1",
      "node": [
        {
          "node-id": "ovsdb://uuid/163724f4-6a70-428a-a8a0-63b2a21f12dd",
          "ovsdb:openvswitch-external-ids": [
            {
              "external-id-key": "system-id",
              "external-id-value": "ecf160af-e78c-4f6b-a005-83a6baa5c979"
            }
          ],
          "ovsdb:connection-info": {
            "local-ip": "<controller-ip>",
            "remote-port": 46731,
            "remote-ip": "<ovs-host-ip>",
            "local-port": 6640
          },
          "ovsdb:ovs-version": "2.3.1-git4750c96",
          "ovsdb:manager-entry": [
            {
              "target": "tcp:10.11.21.7:6640",
              "connected": true,
              "number_of_connections": 1
            }
          ]
        }
      ]
    }
  ]
}
```

Take note of the *node-id* that was created in this case.

Manage Bridges

The OVSDB Southbound Plugin can be used to manage bridges on an OVS host.

This example shows how to add a bridge to the OVSDB node *ovsdb://HOST1*.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest
```

Body:

```
{
  "network-topology:node": [
    {
      "node-id": "ovsdb://HOST1/bridge/brtest",
      "ovsdb:bridge-name": "brtest",
      "ovsdb:protocol-entry": [
        {
          "protocol": "ovsdb:ovsdb-bridge-protocol-openflow-13"
        }
      ],
      "ovsdb:managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb://HOST1']"
```

(continues on next page)

(continued from previous page)

```
    }
  ]
}
```

Notice that the *ovsdb:managed-by* attribute is specified in the command. This indicates the association of the new bridge node with its OVSDB node.

Bridges can be updated. In the following example, OpenDaylight is configured to be the OpenFlow controller for the bridge.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest
```

Body:

```
{
  "network-topology:node": [
        {
          "node-id": "ovsdb://HOST1/bridge/brtest",
             "ovsdb:bridge-name": "brtest",
              "ovsdb:controller-entry": [
                {
                  "target": "tcp:<controller-ip>:6653"
                }
              ],
             "ovsdb:managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb://HOST1']"
        }
    ]
}
```

If the OpenDaylight OpenFlow Plugin is installed, then checking on the OVS host will show that OpenDaylight has successfully connected as the controller for the bridge.

```
$ sudo ovs-vsctl show
    Manager "ptcp:6640"
        is_connected: true
    Bridge brtest
        Controller "tcp:<controller-ip>:6653"
            is_connected: true
        Port brtest
            Interface brtest
                type: internal
    ovs_version: "2.3.1-git4750c96"
```

Query the operational MD-SAL to see how the bridge appears.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest/
```

Result Body:

```
{
  "node": [
    {
      "node-id": "ovsdb://HOST1/bridge/brtest",
      "ovsdb:bridge-name": "brtest",
      "ovsdb:datapath-type": "ovsdb:datapath-type-system",
      "ovsdb:datapath-id": "00:00:da:e9:0c:08:2d:45",
      "ovsdb:managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb://HOST1']",
      "ovsdb:bridge-external-ids": [
        {
          "bridge-external-id-key": "opendaylight-iid",
          "bridge-external-id-value": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb://HOST1/bridge/brtest']"
        }
      ],
      "ovsdb:protocol-entry": [
        {
          "protocol": "ovsdb:ovsdb-bridge-protocol-openflow-13"
        }
      ],
      "ovsdb:bridge-uuid": "080ce9da-101e-452d-94cd-ee8bef8a4b69",
      "ovsdb:controller-entry": [
        {
          "target": "tcp:10.11.21.7:6653",
          "is-connected": true,
          "controller-uuid": "c39b1262-0876-4613-8bfd-c67eec1a991b"
        }
      ],
      "termination-point": [
        {
          "tp-id": "brtest",
          "ovsdb:port-uuid": "c808ae8d-7af2-4323-83c1-e397696dc9c8",
          "ovsdb:ofport": 65534,
          "ovsdb:interface-type": "ovsdb:interface-type-internal",
          "ovsdb:interface-uuid": "49e9417f-4479-4ede-8faf-7c873b8c0413",
          "ovsdb:name": "brtest"
        }
      ]
    }
  ]
}
```


Notice that just like with the OVSDB node, an *opendaylight-iid* has been added to the external-ids column of the bridge since it was created via the configuration MD-SAL.

A bridge node may be deleted as well.

HTTP DELETE:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest
```

Manage Ports

Similarly, ports may be managed by the OVSDB Southbound Plugin.

This example illustrates how a port and various attributes may be created on a bridge.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest/termination-point/
↪testport/
```

Body:

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:options": [
        {
          "ovsdb:option": "remote_ip",
          "ovsdb:value" : "10.10.14.11"
        }
      ],
      "ovsdb:name": "testport",
      "ovsdb:interface-type": "ovsdb:interface-type-vxlan",
      "tp-id": "testport",
      "vlan-tag": "1",
      "trunks": [
        {
          "trunk": "5"
        }
      ],
      "vlan-mode":"access"
    }
  ]
}
```

Ports can be updated - add another VLAN trunk.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest/termination-point/
↪testport/
```

Body:

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:name": "testport",
      "tp-id": "testport",
      "trunks": [
        {
          "trunk": "5"
        },
        {
          "trunk": "500"
        }
```

(continues on next page)

(continued from previous page)

```
      ]
    }
  ]
}
```

Query the operational MD-SAL for the port.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest/termination-point/
↪testport/
```

Result Body:

```
{
  "termination-point": [
    {
      "tp-id": "testport",
      "ovsdb:port-uuid": "b1262110-2a4f-4442-b0df-84faf145488d",
      "ovsdb:options": [
        {
          "option": "remote_ip",
          "value": "10.10.14.11"
        }
      ],
      "ovsdb:port-external-ids": [
        {
          "external-id-key": "opendaylight-iid",
          "external-id-value": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb://HOST1/bridge/brtest']/network-
↪topology:termination-point[network-topology:tp-id='testport']"
        }
      ],
      "ovsdb:interface-type": "ovsdb:interface-type-vxlan",
      "ovsdb:trunks": [
        {
          "trunk": 5
        },
        {
          "trunk": 500
        }
      ],
      "ovsdb:vlan-mode": "access",
      "ovsdb:vlan-tag": 1,
      "ovsdb:interface-uuid": "7cec653b-f407-45a8-baec-7eb36b6791c9",
      "ovsdb:name": "testport",
      "ovsdb:ofport": 1
    }
  ]
}
```

Remember that the OVSDB YANG model includes both OVSDB port and interface table attributes in the termination-point augmentation. Both kinds of attributes can be seen in the examples above. Again, note the creation of an *opendaylight-iid* value in the external-ids column of the port table.

Delete a port.

HTTP DELETE:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbrtest2/termination-point/
↪testport/
```

Overview of QoS and Queue

The OVSDB Southbound Plugin provides the capability of managing the QoS and Queue tables on an OVS host with OpenDaylight configured as the OVSDB manager.

QoS and Queue Tables in OVSDB

The OVSDB includes a QoS and Queue table. Unlike most of the other tables in the OVSDB, except the Open_vSwitch table, the QoS and Queue tables are "root set" tables, which means that entries, or rows, in these tables are not automatically deleted if they can not be reached directly or indirectly from the Open_vSwitch table. This means that QoS entries can exist and be managed independently of whether or not they are referenced in a Port entry. Similarly, Queue entries can be managed independently of whether or not they are referenced by a QoS entry.

Modelling of QoS and Queue Tables in OpenDaylight MD-SAL

Since the QoS and Queue tables are "root set" tables, they are modeled in the OpenDaylight MD-SAL as lists which are part of the attributes of the OVSDB node model.

The MD-SAL QoS and Queue models have an additonal identifier attribute per entry (e.g. "qos-id" or "queue-id") which is not present in the OVSDB schema. This identifier is used by the MD-SAL as a key for referencing the entry. If the entry is created originally from the configuration MD-SAL, then the value of the identifier is whatever is specified by the configuration. If the entry is created on the OVSDB node and received by OpenDaylight in an operational update, then the id will be created in the following format.

```
"queue-id": "queue://<UUID>"
"qos-id": "qos://<UUID>"
```

The UUID in the above identifiers is the actual UUID of the entry in the OVSDB database.

When the QoS or Queue entry is created by the configuration MD-SAL, the identifier will be configured as part of the external-ids column of the entry. This will ensure that the corresponding entry that is created in the operational MD-SAL uses the same identifier.

```
"queues-external-ids": [
  {
    "queues-external-id-key": "opendaylight-queue-id",
    "queues-external-id-value": "QUEUE-1"
  }
]
```

See more in the examples that follow in this section.

The QoS schema in OVSDB currently defines two types of QoS entries.

- linux-htb
- linux-hfsc

These QoS types are defined in the QoS model. Additional types will need to be added to the model in order to be supported. See the examples that folow for how the QoS type is specified in the model.

QoS entries can be configured with addtional attritubes such as "max-rate". These are configured via the *other-config* column of the QoS entry. Refer to OVSDB schema (in the reference section below) for all of the relevant attributes that can be configured. The examples in the rest of this section will demonstrate how the other-config column may be configured.

Similarly, the Queue entries may be configured with additional attributes via the other-config column.

Managing QoS and Queues via Configuration MD-SAL

This section will show some examples on how to manage QoS and Queue entries via the configuration MD-SAL. The examples will be illustrated by using RESTCONF (see QoS and Queue Postman Collection).

A pre-requisite for managing QoS and Queue entries is that the OVS host must be present in the configuration MD-SAL.

For the following examples, the following OVS host is configured.

HTTP POST:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/
```

Body:

```
{
  "node": [
    {
      "node-id": "ovsdb:HOST1",
      "connection-info": {
        "ovsdb:remote-ip": "<ovs-host-ip>",
        "ovsdb:remote-port": "<ovs-host-ovsdb-port>"
      }
    }
  ]
}
```

Where

- <*controller-ip*> is the IP address of the OpenDaylight controller
- <*ovs-host-ip*> is the IP address of the OVS host
- <*ovs-host-ovsdb-port*> is the TCP port of the OVSDB server on the OVS host (e.g. 6640)

This command creates an OVSDB node with the node-id "ovsdb:HOST1". This OVSDB node will be used in the following examples.

QoS and Queue entries can be created and managed without a port, but ultimately, QoS entries are associated with a port in order to use them. For the following examples a test bridge and port will be created.

Create the test bridge.

HTTP PUT

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test
```

Body:

```
{
  "network-topology:node": [
    {
      "node-id": "ovsdb:HOST1/bridge/br-test",
      "ovsdb:bridge-name": "br-test",
      "ovsdb:managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb:HOST1']"
    }
  ]
}
```

Create the test port (which is modeled as a termination point in the OpenDaylight MD-SAL).

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test/termination-point/testport/
```

Body:

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:name": "testport",
      "tp-id": "testport"
    }
  ]
}
```

If all of the previous steps were successful, a query of the operational MD-SAL should look something like the following results. This indicates that the configuration commands have been successfully instantiated on the OVS host.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test
```

Result Body:

```
{
  "node": [
    {
      "node-id": "ovsdb:HOST1/bridge/br-test",
      "ovsdb:bridge-name": "br-test",
      "ovsdb:datapath-type": "ovsdb:datapath-type-system",
      "ovsdb:managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb:HOST1']",
      "ovsdb:datapath-id": "00:00:8e:5d:22:3d:09:49",
      "ovsdb:bridge-external-ids": [
        {
          "bridge-external-id-key": "opendaylight-iid",
          "bridge-external-id-value": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb:HOST1/bridge/br-test']"
        }
```

(continues on next page)

(continued from previous page)

```
      ],
      "ovsdb:bridge-uuid": "3d225d8d-d060-4909-93ef-6f4db58ef7cc",
      "termination-point": [
        {
          "tp-id": "br=-est",
          "ovsdb:port-uuid": "f85f7aa7-4956-40e4-9c94-e6ca2d5cd254",
          "ovsdb:ofport": 65534,
          "ovsdb:interface-type": "ovsdb:interface-type-internal",
          "ovsdb:interface-uuid": "29ff3692-6ed4-4ad7-a077-1edc277ecb1a",
          "ovsdb:name": "br-test"
        },
        {
          "tp-id": "testport",
          "ovsdb:port-uuid": "aa79a8e2-147f-403a-9fa9-6ee5ec276f08",
          "ovsdb:port-external-ids": [
            {
              "external-id-key": "opendaylight-iid",
              "external-id-value": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb:HOST1/bridge/br-test']/network-
↪topology:termination-point[network-topology:tp-id='testport']"
            }
          ],
          "ovsdb:interface-uuid": "e96f282e-882c-41dd-a870-80e6b29136ac",
          "ovsdb:name": "testport"
        }
      ]
    }
  ]
}
```

Create Queue

Create a new Queue in the configuration MD-SAL.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:queues/QUEUE-1/
```

Body:

```
{
  "ovsdb:queues": [
    {
      "queue-id": "QUEUE-1",
      "dscp": 25,
      "queues-other-config": [
        {
          "queue-other-config-key": "max-rate",
          "queue-other-config-value": "3600000"
        }
      ]
    }
  ]
}
```

Query Queue

Now query the operational MD-SAL for the Queue entry.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:queues/QUEUE-1/
```

Result Body:

```
{
  "ovsdb:queues": [
    {
      "queue-id": "QUEUE-1",
      "queues-other-config": [
        {
          "queue-other-config-key": "max-rate",
          "queue-other-config-value": "3600000"
        }
      ],
      "queues-external-ids": [
        {
          "queues-external-id-key": "opendaylight-queue-id",
          "queues-external-id-value": "QUEUE-1"
        }
      ],
      "queue-uuid": "83640357-3596-4877-9527-b472aa854d69",
      "dscp": 25
    }
  ]
}
```

Create QoS

Create a QoS entry. Note that the UUID of the Queue entry, obtained by querying the operational MD-SAL of the Queue entry, is specified in the queue-list of the QoS entry. Queue entries may be added to the QoS entry at the creation of the QoS entry, or by a subsequent update to the QoS entry.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:qos-entries/QOS-1/
```

Body:

```
{
  "ovsdb:qos-entries": [
    {
      "qos-id": "QOS-1",
      "qos-type": "ovsdb:qos-type-linux-htb",
      "qos-other-config": [
        {
          "other-config-key": "max-rate",
          "other-config-value": "4400000"
        }
```

(continues on next page)

(continued from previous page)

```
      ],
      "queue-list": [
        {
          "queue-number": "0",
          "queue-uuid": "83640357-3596-4877-9527-b472aa854d69"
        }
      ]
    }
  ]
}
```

Query QoS

Query the operational MD-SAL for the QoS entry.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:qos-entries/QOS-1/
```

Result Body:

```
{
  "ovsdb:qos-entries": [
    {
      "qos-id": "QOS-1",
      "qos-other-config": [
        {
          "other-config-key": "max-rate",
          "other-config-value": "4400000"
        }
      ],
      "queue-list": [
        {
          "queue-number": 0,
          "queue-uuid": "83640357-3596-4877-9527-b472aa854d69"
        }
      ],
      "qos-type": "ovsdb:qos-type-linux-htb",
      "qos-external-ids": [
        {
          "qos-external-id-key": "opendaylight-qos-id",
          "qos-external-id-value": "QOS-1"
        }
      ],
      "qos-uuid": "90ba9c60-3aac-499d-9be7-555f19a6bb31"
    }
  ]
}
```

Add QoS to a Port

Update the termination point entry to include the UUID of the QoS entry, obtained by querying the operational MD-SAL, to associate a QoS entry with a port.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test/termination-point/testport/
```

Body:

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:name": "testport",
      "tp-id": "testport",
      "qos": "90ba9c60-3aac-499d-9be7-555f19a6bb31"
    }
  ]
}
```

Query the Port

Query the operational MD-SAL to see how the QoS entry appears in the termination point model.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test/termination-point/testport/
```

Result Body:

```
{
  "termination-point": [
    {
      "tp-id": "testport",
      "ovsdb:port-uuid": "aa79a8e2-147f-403a-9fa9-6ee5ec276f08",
      "ovsdb:port-external-ids": [
        {
          "external-id-key": "opendaylight-iid",
          "external-id-value": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb:HOST1/bridge/br-test']/network-
↪topology:termination-point[network-topology:tp-id='testport']"
        }
      ],
      "ovsdb:qos": "90ba9c60-3aac-499d-9be7-555f19a6bb31",
      "ovsdb:interface-uuid": "e96f282e-882c-41dd-a870-80e6b29136ac",
      "ovsdb:name": "testport"
    }
  ]
}
```

Query the OVSDB Node

Query the operational MD-SAL for the OVS host to see how the QoS and Queue entries appear as lists in the OVS node model.

HTTP GET:

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/
```

Result Body (edited to only show information relevant to the QoS and Queue entries):

```
{
  "node": [
    {
      "node-id": "ovsdb:HOST1",
      <content edited out>
      "ovsdb:queues": [
        {
          "queue-id": "QUEUE-1",
          "queues-other-config": [
            {
              "queue-other-config-key": "max-rate",
              "queue-other-config-value": "3600000"
            }
          ],
          "queues-external-ids": [
            {
              "queues-external-id-key": "opendaylight-queue-id",
              "queues-external-id-value": "QUEUE-1"
            }
          ],
          "queue-uuid": "83640357-3596-4877-9527-b472aa854d69",
          "dscp": 25
        }
      ],
      "ovsdb:qos-entries": [
        {
          "qos-id": "QOS-1",
          "qos-other-config": [
            {
              "other-config-key": "max-rate",
              "other-config-value": "4400000"
            }
          ],
          "queue-list": [
            {
              "queue-number": 0,
              "queue-uuid": "83640357-3596-4877-9527-b472aa854d69"
            }
          ],
          "qos-type": "ovsdb:qos-type-linux-htb",
          "qos-external-ids": [
            {
              "qos-external-id-key": "opendaylight-qos-id",
              "qos-external-id-value": "QOS-1"
            }
          ],
          "qos-uuid": "90ba9c60-3aac-499d-9be7-555f19a6bb31"
        }
      ]
      <content edited out>
    }
  ]
}
```

Remove QoS from a Port

This example removes a QoS entry from the termination point and associated port. Note that this is a PUT command on the termination point with the QoS attribute absent. Other attributes of the termination point should be included in the body of the command so that they are not inadvertantly removed.

HTTP PUT:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fbr-test/termination-point/testport/
```

Body:

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:name": "testport",
      "tp-id": "testport"
    }
  ]
}
```

Remove a Queue from QoS

This example removes the specific Queue entry from the queue list in the QoS entry. The queue entry is specified by the queue number, which is "0" in this example.

HTTP DELETE:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:qos-entries/QOS-1/queue-list/0/
```

Remove Queue

Once all references to a specific queue entry have been removed from QoS entries, the Queue itself can be removed.

HTTP DELETE:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:queues/QUEUE-1/
```

Remove QoS

The QoS entry may be removed when it is no longer referenced by any ports.

HTTP DELETE:

```
http://<controller-ip>:8181/restconf/config/network-topology:network-topology/
↪topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:qos-entries/QOS-1/
```

References

Openvswitch schema

OVSDB and Netvirt Postman Collection

1.1.3 OVSDB Hardware VTEP SouthBound Plugin

Overview

Hwvtepsouthbound plugin is used to configure a hardware VTEP which implements hardware ovsdb schema. This page will show how to use RESTConf API of hwvtepsouthbound. There are two ways to connect to ODL:

switch initiates connection..

Both will be introduced respectively.

User Initiates Connection

Prerequisite

Configure the hwvtep device/node to listen for the tcp connection in passive mode. In addition, management IP and tunnel source IP are also configured. After all this configuration is done, a physical switch is created automatically by the hwvtep node.

Connect to a hwvtep device/node

Send below Restconf request if you want to initiate the connection to a hwvtep node from the controller, where listening IP and port of hwvtep device/node are provided.

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/

```
{
 "network-topology:node": [
        {
            "node-id": "hwvtep://192.168.1.115:6640",
            "hwvtep:connection-info":
            {
                "hwvtep:remote-port": 6640,
                "hwvtep:remote-ip": "192.168.1.115"
            }
        }
   ]
}
```

Please replace *odl* in the URL with the IP address of your OpenDaylight controller and change *192.168.1.115* to your hwvtep node IP.

NOTE: The format of node-id is fixed. It will be one of the two:

User initiates connection from ODL:

```
hwvtep://ip:port
```

Switch initiates connection:

```
hwvtep://uuid/<uuid of switch>
```

The reason for using UUID is that we can distinguish between multiple switches if they are behind a NAT.

After this request is completed successfully, we can get the physical switch from the operational data store.

REST API: GET http://odl:8181/restconf/operational/network-topology:network-topology/topology/hwvtep:1/node/hwvtep:%2F%2F192.168.1.115:6640

There is no body in this request.

The response of the request is:


```
{
   "node": [
         {
           "node-id": "hwvtep://192.168.1.115:6640",
           "hwvtep:switches": [
             {
               "switch-ref": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640/physicalswitch/
↪br0']"
             }
           ],
           "hwvtep:connection-info": {
             "local-ip": "192.168.92.145",
             "local-port": 47802,
             "remote-port": 6640,
             "remote-ip": "192.168.1.115"
           }
         },
         {
           "node-id": "hwvtep://192.168.1.115:6640/physicalswitch/br0",
           "hwvtep:management-ips": [
             {
               "management-ips-key": "192.168.1.115"
             }
           ],
           "hwvtep:physical-switch-uuid": "37eb5abd-a6a3-4aba-9952-a4d301bdf371",
           "hwvtep:managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640']",
           "hwvtep:hwvtep-node-description": "",
           "hwvtep:tunnel-ips": [
             {
               "tunnel-ips-key": "192.168.1.115"
             }
           ],
           "hwvtep:hwvtep-node-name": "br0"
         }
       ]
}
```


If there is a physical switch which has already been created by manual configuration, we can get the node-id of the physical switch from this response, which is presented in "swith-ref". If the switch does not exist, we need to create the physical switch. Currently, most hwvtep devices do not support running multiple switches.

Create a physical switch

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/

request body:

```
{
 "network-topology:node": [
       {
           "node-id": "hwvtep://192.168.1.115:6640/physicalswitch/br0",
           "hwvtep-node-name": "ps0",
           "hwvtep-node-description": "",
           "management-ips": [
             {
               "management-ips-key": "192.168.1.115"
             }
           ],
           "tunnel-ips": [
             {
               "tunnel-ips-key": "192.168.1.115"
             }
           ],
           "managed-by": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640']"
       }
   ]
}
```

Note: "managed-by" must provided by user. We can get its value after the step *Connect to a hwvtep device/node* since the node-id of hwvtep device is provided by user. "managed-by" is a reference typed of instance identifier. Though the instance identifier is a little complicated for RestConf, the primary user of hwvtepsouthbound plugin will be provider-type code such as NetVirt and the instance identifier is much easier to write code for.

Create a logical switch

Creating a logical switch is effectively creating a logical network. For VxLAN, it is a tunnel network with the same VNI.

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/node/hwvtep:%2F%2F192.168.1.115:6640

request body:

```
{
 "logical-switches": [
       {
           "hwvtep-node-name": "ls0",
           "hwvtep-node-description": "",
           "tunnel-key": "10000"
        }
   ]
}
```

Create a physical locator

After the VXLAN network is ready, we will add VTEPs to it. A VTEP is described by a physical locator.

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/node/hwvtep:%2F%2F192.168.1.115:6640

request body:

```
{
 "termination-point": [
      {
          "tp-id": "vxlan_over_ipv4:192.168.0.116",
          "encapsulation-type": "encapsulation-type-vxlan-over-ipv4",
          "dst-ip": "192.168.0.116"
          }
     ]
}
```

The "tp-id" of locator is "{encapsualation-type}: {dst-ip}".

Note: As far as we know, the OVSDB database does not allow the insertion of a new locator alone. So, no locator is inserted after this request is sent. We will trigger off the creation until other entity refer to it, such as remote-mcast-macs.

Create a remote-mcast-macs entry

After adding a physical locator to a logical switch, we need to create a remote-mcast-macs entry to handle unknown traffic.

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/node/hwvtep:%2F%2F192.168.1.115:6640

request body:

```
{
 "remote-mcast-macs": [
       {
           "mac-entry-key": "00:00:00:00:00:00",
           "logical-switch-ref": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640']/
↪hwvtep:logical-switches[hwvtep:hwvtep-node-name='ls0']",
           "locator-set": [
                 {
                       "locator-ref": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://219.141.189.115:6640']/network-
↪topology:termination-point[network-topology:tp-id='vxlan_over_ipv4:192.168.0.116']"
                 }
           ]
       }
   ]
}
```

The physical locator *vxlan_over_ipv4:192.168.0.116* is just created in "Create a physical locator". It should be noted that list "locator-set" is immutable, that is, we must provide a set of "locator-ref" as a whole.

Note: "00:00:00:00:00:00" stands for "unknown-dst" since the type of mac-entry-key is yang:mac and does not accept "unknown-dst".

Create a physical port

Now we add a physical port into the physical switch "hwvtep://192.168.1.115:6640/physicalswitch/br0". The port is attached with a physical server or an L2 network and with the vlan 100.

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/node/hwvtep:%2F%2F192.168.1.115:6640%2Fphysicalswitch%2Fbr0

```
{
 "network-topology:termination-point": [
       {
           "tp-id": "port0",
           "hwvtep-node-name": "port0",
           "hwvtep-node-description": "",
           "vlan-bindings": [
               {
                 "vlan-id-key": "100",
                 "logical-switch-ref": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640']/
↪hwvtep:logical-switches[hwvtep:hwvtep-node-name='ls0']"
               }
         ]
       }
   ]
}
```

At this point, we have completed the basic configuration.

Typically, hwvtep devices learn local MAC addresses automatically. But they also support getting MAC address entries from ODL.

Create a local-mcast-macs entry

It is similar to *Create a remote-mcast-macs entry*.

Create a remote-ucast-macs

REST API: POST http://odl:8181/restconf/config/network-topology:network-topology/topology/hwvtep:1/node/hwvtep:%2F%2F192.168.1.115:6640

```
request body:
```

```
{
 "remote-ucast-macs": [
       {
           "mac-entry-key": "11:11:11:11:11:11",
           "logical-switch-ref": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640']/
↪hwvtep:logical-switches[hwvtep:hwvtep-node-name='ls0']",
```

(continues on next page)

(continued from previous page)

```
            "ipaddr": "1.1.1.1",
            "locator-ref": "/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='hwvtep:1']/network-
↪topology:node[network-topology:node-id='hwvtep://192.168.1.115:6640']/network-
↪topology:termination-point[network-topology:tp-id='vxlan_over_ipv4:192.168.0.116']"
        }
    ]
}
```

Create a local-ucast-macs entry

This is similar to *Create a remote-ucast-macs*.

Switch Initiates Connection

We do not need to connect to a hwvtep device/node when the switch initiates the connection. After switches connect to ODL successfully, we get the node-id's of switches by reading the operational data store. Once the node-id of a hwvtep device is received, the remaining steps are the same as when the user initiates the connection.

References

https://wiki.opendaylight.org/view/User_talk:Pzhang

CHAPTER 2

OVSDB Developer Guide

2.1 OVSDB Integration

The Open vSwitch database (OVSDB) Southbound Plugin component for OpenDaylight implements the OVSDB RFC 7047 management protocol that allows the southbound configuration of switches that support OVSDB. The component comprises a library and a plugin. The OVSDB protocol uses JSON-RPC calls to manipulate a physical or virtual switch that supports OVSDB. Many vendors support OVSDB on various hardware platforms. The OpenDaylight controller uses the library project to interact with an OVS instance.

Note: Read the OVSDB User Guide before you begin development.

2.1.1 OpenDaylight OVSDB southbound plugin architecture and design

OpenVSwitch (OVS) is generally accepted as the unofficial standard for Virtual Switching in the Open hypervisor based solutions. Every other Virtual Switch implementation, properietery or otherwise, uses OVS in some form. For information on OVS, see Open vSwitch.

In Software Defined Networking (SDN), controllers and applications interact using two channels: OpenFlow and OVSDB. OpenFlow addresses the forwarding-side of the OVS functionality. OVSDB, on the other hand, addresses the management-plane. A simple and concise overview of Open Virtual Switch Database(OVSDB) is available at: http://networkstatic.net/getting-started-ovsdb/

Overview of OpenDaylight Controller architecture

The OpenDaylight controller platform is designed as a highly modular and plugin based middleware that serves various network applications in a variety of use-cases. The modularity is achieved through the Java OSGi framework. The controller consists of many Java OSGi bundles that work together to provide the required controller functionalities.

The bundles can be placed in the following broad categories:

- Network Service Functional Modules (Examples: Topology Manager, Inventory Manager, Forwarding Rules Manager,and others)
- NorthBound API Modules (Examples: Topology APIs, Bridge Domain APIs, Neutron APIs, Connection Manager APIs, and others)
- Service Abstraction Layer(SAL)- (Inventory Services, DataPath Services, Topology Services, Network Config, and others)
- SouthBound Plugins (OpenFlow Plugin, OVSDB Plugin, OpenDove Plugin, and others)
- Application Modules (Simple Forwarding, Load Balancer)

Each layer of the Controller architecture performs specified tasks, and hence aids in modularity. While the Northbound API layer addresses all the REST-Based application needs, the SAL layer takes care of abstracting the SouthBound plugin protocol specifics from the Network Service functions.

Each of the SouthBound Plugins serves a different purpose, with some overlapping. For example, the OpenFlow plugin might serve the Data-Plane needs of an OVS element, while the OVSDB plugin can serve the management plane needs of the same OVS element. As the OpenFlow Plugin talks OpenFlow protocol with the OVS element, the OVSDB plugin will use OVSDB schema over JSON-RPC transport.

2.1.2 OVSDB southbound plugin

The Open vSwitch Database Management Protocol-draft-02 and Open vSwitch Manual provide theoretical information about OVSDB. The OVSDB protocol draft is generic enough to lay the groundwork on Wire Protocol and Database Operations, and the OVS Manual currently covers 13 tables leaving space for future OVS expansion, and vendor expansions on proprietary implementations. The OVSDB Protocol is a database records transport protocol using JSON RPC1.0. For information on the protocol structure, see Getting Started with OVSDB. The OpenDaylight OVSDB southbound plugin consists of one or more OSGi bundles addressing the following services or functionalities:

- Connection Service - Based on Netty
- Network Configuration Service
- Bidirectional JSON-RPC Library
- OVSDB Schema definitions and Object mappers
- Overlay Tunnel management
- OVSDB to OpenFlow plugin mapping service
- Inventory Service

2.1.3 Connection service

One of the primary services that most southbound plugins provide in OpenDaylight a Connection Service. The service provides protocol specific connectivity to network elements, and supports the connectivity management services as specified by the OpenDaylight Connection Manager. The connectivity services include:

- Connection to a specified element given IP-address, L4-port, and other connectivity options (such as authentication,. . .)
- Disconnection from an element
- Handling Cluster Mode change notifications to support the OpenDaylight Clustering/High-Availability feature

2.1.4 Network Configuration Service

The goal of the OpenDaylight Network Configuration services is to provide complete management plane solutions needed to successfully install, configure, and deploy the various SDN based network services. These are generic services which can be implemented in part or full by any south-bound protocol plugin. The south-bound plugins can be either of the following:

- The new network virtualization protocol plugins such as OVSDB JSON-RPC
- The traditional management protocols such as SNMP or any others in the middle.

The above definition, and more information on Network Configuration Services, is available at : https://wiki.opendaylight.org/view/OpenDaylight_Controller:NetworkConfigurationServices

Bidirectional JSON-RPC library

The OVSDB plugin implements a Bidirectional JSON-RPC library. It is easy to design the library as a module that manages the Netty connection towards the Element.

The main responsibilities of this Library are:

- Demarshal and marshal JSON Strings to JSON objects
- Demarshal and marshal JSON Strings from and to the Network Element.

OVSDB Schema definitions and Object mappers

The OVSDB Schema definitions and Object Mapping layer sits above the JSON-RPC library. It maps the generic JSON objects to OVSDB schema POJOs (Plain Old Java Object) and vice-versa. This layer mostly provides the Java Object definition for the corresponding OVSDB schema (13 of them) and also will provide much more friendly API abstractions on top of these object data. This helps in hiding the JSON semantics from the functional modules such as Configuration Service and Tunnel management.

On the demarshaling side the mapping logic differentiates the Request and Response messages as follows :

- Request messages are mapped by its "method"
- Response messages are mapped by their IDs which were originally populated by the Request message. The JSON semantics of these OVSDB schema is quite complex. The following figures summarize two of the end-to-end scenarios:

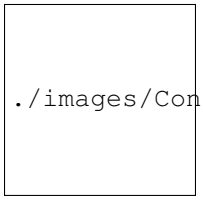

Fig. 1: End-to-end handling of a Create Bridge request

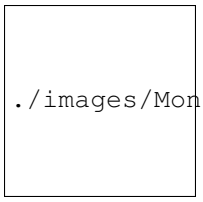

Fig. 2: End-to-end handling of a monitor response

Overlay tunnel management

Network Virtualization using OVS is achieved through Overlay Tunnels. The actual Type of the Tunnel may be GRE, VXLAN, or STT. The differences in the encapsulation and configuration decide the tunnel types. Establishing a tunnel using configuration service requires just the sending of OVSDB messages towards the ovsdb-server. However, the scaling issues that would arise on the state management at the data-plane (using OpenFlow) can get challenging. Also, this module can assist in various optimizations in the presence of Gateways. It can also help in providing Service guarantees for the VMs using these overlays with the help of underlay orchestration.

OVSDB to OpenFlow plugin mapping service

The connect() of the ConnectionService would result in a Node that represents an ovsdb-server. The CreateBridgeDomain() Configuration on the above Node would result in creating an OVS bridge. This OVS Bridge is an OpenFlow Agent for the OpenDaylight OpenFlow plugin with its own Node represented as (example) OF|xxxx.yyyy.zzzz. Without any help from the OVSDB plugin, the Node Mapping Service of the Controller platform would not be able to map the following:

```
{OVSDB_NODE + BRIDGE_IDENTFIER} <---> {OF_NODE}.
```

Without such mapping, it would be extremely difficult for the applications to manage and maintain such nodes. This Mapping Service provided by the OVSDB plugin would essentially help in providing more value added services to the orchestration layers that sit atop the Northbound APIs (such as OpenStack).

2.1.5 OVSDB: New features

Schema independent library

The OVS connection is a node which can have multiple databases. Each database is represented by a schema. A single connection can have multiple schemas. OSVDB supports multiple schemas. Currently, these are two schemas available in the OVSDB, but there is no restriction on the number of schemas. Owing to the Northbound v3 API, no code changes in ODL are needed for supporting additional schemas.

Schemas:

- openvswitch : Schema wrapper that represents http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf
- hardwarevtep: Schema wrapper that represents http://openvswitch.org/docs/vtep.5.pdf

2.2 OVSDB Library Developer Guide

2.2.1 Overview

The OVSDB library manages the Netty connections to network nodes and handles bidirectional JSON-RPC messages. It not only provides OVSDB protocol functionality to OpenDaylight OVSDB plugin but also can be used as standalone JAVA library for OVSDB protocol.

The main responsibilities of OVSDB library include:

- Manage connections to peers
- Marshal and unmarshal JSON Strings to JSON objects.
- Marshal and unmarshal JSON Strings from and to the Network Element.

2.2.2 Connection Service

The OVSDB library provides connection management through the OvsdbConnection interface. The OvsdbConnection interface provides OVSDB connection management APIs which include both active and passive connections. From the library perspective, active OVSDB connections are initiated from the controller to OVS nodes while passive OVSDB connections are initiated from OVS nodes to the controller. In the active connection scenario an application needs to provide the IP address and listening port of OVS nodes to the library management API. On the other hand, the library management API only requires the info of the controller listening port in the passive connection scenario.

For a passive connection scenario, the library also provides a connection event listener through the OvsdbConnectionListener interface. The listener interface has connected() and disconnected() methods to notify an application when a new passive connection is established or an existing connection is terminated.

2.2.3 SSL Connection

In addition to a regular TCP connection, the OvsdbConnection interface also provides a connection management API for an SSL connection. To start an OVSDB connection with SSL, an application will need to provide a Java SSLContext object to the management API. There are different ways to create a Java SSLContext, but in most cases a Java KeyStore with certificate and private key provided by the application is required. Detailed steps about how to create a Java SSLContext is out of the scope of this document and can be found in the Java documentation for JAVA Class SSlContext.

In the active connection scenario, the library uses the given SSLContext to create a Java SSLEngine and configures the SSL engine with the client mode for SSL handshaking. Normally clients are not required to authenticate themselves.

In the passive connection scenario, the library uses the given SSLContext to create a Java SSLEngine which will operate in server mode for SSL handshaking. For security reasons, the SSLv3 protocol and some cipher suites are disabled. Currently the OVSDB server only supports the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite and the following protocols: SSLv2Hello, TLSv1, TLSv1.1, TLSv1.2.

The SSL engine is also configured to operate on two-way authentication mode for passive connection scenarios, i.e., the OVSDB server (controller) will authenticate clients (OVS nodes) and clients (OVS nodes) are also required to authenticate the server (controller). In the two-way authentication mode, an application should keep a trust manager to store the certificates of trusted clients and initialize a Java SSLContext with this trust manager. Thus during the SSL handshaking process the OVSDB server (controller) can use the trust manager to verify clients and only accept connection requests from trusted clients. On the other hand, users should also configure OVS nodes to authenticate the controller. Open vSwitch already supports this functionality in the ovsdb-server command with option `--ca-cert=cacert.pem` and `--bootstrap-ca-cert=cacert.pem`. On the OVS node, a user can use the option `--ca-cert=cacert.pem` to specify a controller certificate directly and the node will only

allow connections to the controller with the specified certificate. If the OVS node runs ovsdb-server with option `--bootstrap-ca-cert=cacert.pem`, it will authenticate the controller with the specified certificate cacert.pem. If the certificate file doesn't exist, it will attempt to obtain a certificate from the peer (controller) on its first SSL connection and save it to the named PEM file `cacert.pem`. Here is an example of ovsdb-server with `--bootstrap-ca-cert=cacert.pem` option:

```
ovsdb-server --pidfile --detach --log-file --remote punix:/var/run/
openvswitch/db.sock --remote=db:hardware_vtep,Global,managers --private-key=/
etc/openvswitch/ovsclient-privkey.pem -- certificate=/etc/openvswitch/
ovsclient-cert.pem --bootstrap-ca-cert=/etc/openvswitch/vswitchd.cacert
```

2.2.4 OVSDB protocol transactions

The OVSDB protocol defines the RPC transaction methods in RFC 7047. The following RPC methods are supported in OVSDB protocol:

- List databases
- Get schema
- Transact
- Cancel
- Monitor
- Update notification
- Monitor cancellation
- Lock operations
- Locked notification
- Stolen notification
- Echo

According to RFC 7047, an OVSDB server must implement all methods, and an OVSDB client is only required to implement the "Echo" method and otherwise free to implement whichever methods suit its needs. However, the OVSDB library currently doesn't support all RPC methods. For the "Echo" method, the library can handle "Echo" messages from a peer and send a JSON response message back, but the library doesn't support actively sending an "Echo" JSON request to a peer. Other unsupported RPC methods are listed below:

- Cancel
- Lock operations
- Locked notification
- Stolen notification

In the OVSDB library the RPC methods are defined in the Java interface OvsdbRPC. The library also provides a high-level interface OvsdbClient as the main interface to interact with peers through the OVSDB protocol. In the passive connection scenario, each connection will have a corresponding OvsdbClient object, and the application can obtain the OvsdbClient object through connection listener callback methods. In other words, if the application implements the OvsdbConnectionListener interface, it will get notifications of connection status changes with the corresponding OvsdbClient object of that connection.

2.2.5 OVSDB database operations

RFC 7047 also defines database operations, such as insert, delete, and update, to be performed as part of a "transact" RPC request. The OVSDB library defines the data operations in Operations.java and provides the TransactionBuilder class to help build "transact" RPC requests. To build a JSON-RPC transact request message, the application can obtain the TransactionBuilder object through a transactBuilder() method in the OvsdbClient interface.

The TransactionBuilder class provides the following methods to help build transactions:

- getOperations(): Get the list of operations in this transaction.
- add(): Add data operation to this transaction.
- build(): Return the list of operations in this transaction. This is the same as the getOperations() method.
- execute(): Send the JSON RPC transaction to peer.
- getDatabaseSchema(): Get the database schema of this transaction.

If the application wants to build and send a "transact" RPC request to modify OVSDB tables on a peer, it can take the following steps:

1. Statically import parameter "op" in Operations.java

   ```
   import static org.opendaylight.ovsdb.lib.operations.Operations.op;
   ```

2. Obtain transaction builder through transacBuilder() method in OvsdbClient:

   ```
   TransactionBuilder transactionBuilder = ovsdbClient.transactionBuilder(dbSchema);
   ```

3. Add operations to transaction builder:

   ```
   transactionBuilder.add(op.insert(schema, row));
   ```

4. Send transaction to peer and get JSON RPC response:

   ```
   operationResults = transactionBuilder.execute().get();
   ```

Note: Although the "select" operation is supported in the OVSDB library, the library implementation is a little different from RFC 7047. In RFC 7047, section 5.2.2 describes the "select" operation as follows:

"The "rows" member of the result is an array of objects. Each object corresponds to a matching row, with each column specified in "columns" as a member, the column's name as the member name, and its value as the member value. If "columns" is not specified, all the table's columns are included (including the internally generated "_uuid" and "_version" columns)."

The OVSDB library implementation always requires the column's name in the "columns" field of a JSON message. If the "columns" field is not specified, none of the table's columns are included. If the application wants to get the table entry with all columns, it needs to specify all the columns' names in the "columns" field.

2.2.6 Reference Documentation

RFC 7047 The Open vSwitch Databse Management Protocol https://tools.ietf.org/html/rfc7047

2.3 OVSDB MD-SAL Southbound Plugin Developer Guide

2.3.1 Overview

The Open vSwitch Database (OVSDB) Model Driven Service Abstraction Layer (MD-SAL) Southbound Plugin provides an MD-SAL based interface to Open vSwitch systems. This is done by augmenting the MD-SAL topology node with a YANG model which replicates some (but not all) of the Open vSwitch schema.

2.3.2 OVSDB MD-SAL Southbound Plugin Architecture and Operation

The architecture and operation of the OVSDB MD-SAL Southbound plugin is illustrated in the following set of diagrams.

Connecting to an OVSDB Node

An OVSDB node is a system which is running the OVS software and is capable of being managed by an OVSDB manager. The OVSDB MD-SAL Southbound plugin in OpenDaylight is capable of operating as an OVSDB manager. Depending on the configuration of the OVSDB node, the connection of the OVSDB manager can be active or passive.

Active OVSDB Node Manager Workflow

An active OVSDB node manager connection is made when OpenDaylight initiates the connection to the OVSDB node. In order for this to work, you must configure the OVSDB node to listen on a TCP port for the connection (i.e. OpenDaylight is active and the OVSDB node is passive). This option can be configured on the OVSDB node using the following command:

```
ovs-vsctl set-manager ptcp:6640
```

The following diagram illustrates the sequence of events which occur when OpenDaylight initiates an active OVSDB manager connection to an OVSDB node.

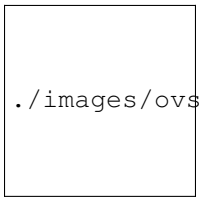

Fig. 3: Active OVSDB Manager Connection

Step 1 Create an OVSDB node by using RESTCONF or an OpenDaylight plugin. The OVSDB node is listed under the OVSDB topology node.

Step 2 Add the OVSDB node to the OVSDB MD-SAL southbound configuration datastore. The OVSDB southbound provider is registered to listen for data change events on the portion of the MD-SAL topology data store which contains the OVSDB southbound topology node augmentations. The addition of an OVSDB node causes an event which is received by the OVSDB Southbound provider.

Step 3 The OVSDB Southbound provider initiates a connection to the OVSDB node using the connection information provided in the configuration OVSDB node (i.e. IP address and TCP port number).

Step 4 The OVSDB Southbound provider adds the OVSDB node to the OVSDB MD-SAL operational data store. The operational data store contains OVSDB node objects which represent active connections to OVSDB nodes.

Step 5 The OVSDB Southbound provider requests the schema and databases which are supported by the OVSDB node.

Step 6 The OVSDB Southbound provider uses the database and schema information to construct a monitor request which causes the OVSDB node to send the controller any updates made to the OVSDB databases on the OVSDB node.

Passive OVSDB Node Manager Workflow

A passive OVSDB node connection to OpenDaylight is made when the OVSDB node initiates the connection to OpenDaylight. In order for this to work, you must configure the OVSDB node to connect to the IP address and OVSDB port on which OpenDaylight is listening. This option can be configured on the OVSDB node using the following command:

```
ovs-vsctl set-manager tcp:<IP address>:6640
```

The following diagram illustrates the sequence of events which occur when an OVSDB node connects to OpenDaylight.

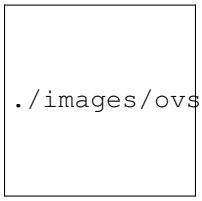

Fig. 4: Passive OVSDB Manager Connection

Step 1 The OVSDB node initiates a connection to OpenDaylight.

Step 2 The OVSDB Southbound provider adds the OVSDB node to the OVSDB MD-SAL operational data store. The operational data store contains OVSDB node objects which represent active connections to OVSDB nodes.

Step 3 The OVSDB Southbound provider requests the schema and databases which are supported by the OVSDB node.

Step 4 The OVSDB Southbound provider uses the database and schema information to construct a monitor request which causes the OVSDB node to send back any updates which have been made to the OVSDB databases on the OVSDB node.

OVSDB Node ID in the Southbound Operational MD-SAL

When OpenDaylight initiates an active connection to an OVSDB node, it writes an external-id to the Open_vSwitch table on the OVSDB node. The external-id is an OpenDaylight instance identifier which identifies the OVSDB topology node which has just been created. Here is an example showing the value of the *opendaylight-iid* entry in the external-ids column of the Open_vSwitch table where the node-id of the OVSDB node is *ovsdb:HOST1*.

```
$ ovs-vsctl list open_vswitch
...
external_ids        : {opendaylight-iid="/network-topology:network-topology/network-
↪topology:topology[network-topology:topology-id='ovsdb:1']/network-
↪topology:node[network-topology:node-id='ovsdb:HOST1']"}
...
```

The *opendaylight-iid* entry in the external-ids column of the Open_vSwitch table causes the OVSDB node to have same node-id in the operational MD-SAL datastore as in the configuration MD-SAL datastore. This holds true if the OVSDB node manager settings are subsequently changed so that a passive OVSDB manager connection is made.

If there is no *opendaylight-iid* entry in the external-ids column and a passive OVSDB manager connection is made, then the node-id of the OVSDB node in the operational MD-SAL datastore will be constructed using the UUID of the Open_vSwitch table as follows.

```
"node-id": "ovsdb://uuid/b8dc0bfb-d22b-4938-a2e8-b0084d7bd8c1"
```

The *opendaylight-iid* entry can be removed from the Open_vSwitch table using the following command.

```
$ sudo ovs-vsctl remove open_vswitch . external-id "opendaylight-iid"
```

OVSDB Changes by using OVSDB Southbound Config MD-SAL

After the connection has been made to an OVSDB node, you can make changes to the OVSDB node by using the OVSDB Southbound Config MD-SAL. You can make CRUD operations by using the RESTCONF interface or by a plugin using the MD-SAL APIs. The following diagram illustrates the high-level flow of events.

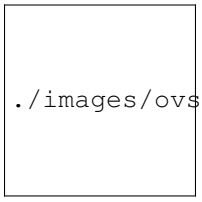

Fig. 5: OVSDB Changes by using the Southbound Config MD-SAL

Step 1 A change to the OVSDB Southbound Config MD-SAL is made. Changes include adding or deleting bridges and ports, or setting attributes of OVSDB nodes, bridges or ports.

Step 2 The OVSDB Southbound provider receives notification of the changes made to the OVSDB Southbound Config MD-SAL data store.

Step 3 As appropriate, OVSDB transactions are constructed and transmitted to the OVSDB node to update the OVSDB database on the OVSDB node.

Step 4 The OVSDB node sends update messages to the OVSDB Southbound provider to indicate the changes made to the OVSDB nodes database.

Step 5 The OVSDB Southbound provider maps the changes received from the OVSDB node into corresponding changes made to the OVSDB Southbound Operational MD-SAL data store.

Detecting changes in OVSDB coming from outside OpenDaylight

Changes to the OVSDB nodes database may also occur independently of OpenDaylight. OpenDaylight also receives notifications for these events and updates the Southbound operational MD-SAL. The following diagram illustrates the sequence of events.

Step 1 Changes are made to the OVSDB node outside of OpenDaylight (e.g. ovs-vsctl).

Step 2 The OVSDB node constructs update messages to inform OpenDaylight of the changes made to its databases.

Step 3 The OVSDB Southbound provider maps the OVSDB database changes to corresponding changes in the OVSDB Southbound operational MD-SAL data store.

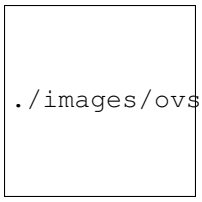

Fig. 6: OVSDB Changes made directly on the OVSDB node

OVSDB Model

The OVSDB Southbound MD-SAL operates using a YANG model which is based on the abstract topology node model found in the network topology model.

The augmentations for the OVSDB Southbound MD-SAL are defined in the ovsdb.yang file.

There are three augmentations:

ovsdb-node-augmentation This augments the topology node and maps primarily to the Open_vSwitch table of the OVSDB schema. It contains the following attributes.

- **connection-info** - holds the local and remote IP address and TCP port numbers for the OpenDaylight to OVSDB node connections
- **db-version** - version of the OVSDB database
- **ovs-version** - version of OVS
- **list managed-node-entry** - a list of references to ovsdb-bridge-augmentation nodes, which are the OVS bridges managed by this OVSDB node
- **list datapath-type-entry** - a list of the datapath types supported by the OVSDB node (e.g. *system*, *netdev*) - depends on newer OVS versions
- **list interface-type-entry** - a list of the interface types supported by the OVSDB node (e.g. *internal*, *vxlan*, *gre*, *dpdk*, etc.) - depends on newer OVS verions
- **list openvswitch-external-ids** - a list of the key/value pairs in the Open_vSwitch table external_ids column
- **list openvswitch-other-config** - a list of the key/value pairs in the Open_vSwitch table other_config column

ovsdb-bridge-augmentation This augments the topology node and maps to an specific bridge in the OVSDB bridge table of the associated OVSDB node. It contains the following attributes.

- **bridge-uuid** - UUID of the OVSDB bridge
- **bridge-name** - name of the OVSDB bridge
- **bridge-openflow-node-ref** - a reference (instance-identifier) of the OpenFlow node associated with this bridge
- **list protocol-entry** - the version of OpenFlow protocol to use with the OpenFlow controller
- **list controller-entry** - a list of controller-uuid and is-connected status of the OpenFlow controllers associated with this bridge
- **datapath-id** - the datapath ID associated with this bridge on the OVSDB node
- **datapath-type** - the datapath type of this bridge
- **fail-mode** - the OVSDB fail mode setting of this bridge
- **flow-node** - a reference to the flow node corresponding to this bridge

- **managed-by** - a reference to the ovsdb-node-augmentation (OVSDB node) that is managing this bridge
- **list bridge-external-ids** - a list of the key/value pairs in the bridge table external_ids column for this bridge
- **list bridge-other-configs** - a list of the key/value pairs in the bridge table other_config column for this bridge

ovsdb-termination-point-augmentation This augments the topology termination point model. The OVSDB Southbound MD-SAL uses this model to represent both the OVSDB port and OVSDB interface for a given port/interface in the OVSDB schema. It contains the following attributes.

- **port-uuid** - UUID of an OVSDB port row
- **interface-uuid** - UUID of an OVSDB interface row
- **name** - name of the port
- **interface-type** - the interface type
- **list options** - a list of port options
- **ofport** - the OpenFlow port number of the interface
- **ofport_request** - the requested OpenFlow port number for the interface
- **vlan-tag** - the VLAN tag value
- **list trunks** - list of VLAN tag values for trunk mode
- **vlan-mode** - the VLAN mode (e.g. access, native-tagged, native-untagged, trunk)
- **list port-external-ids** - a list of the key/value pairs in the port table external_ids column for this port
- **list interface-external-ids** - a list of the key/value pairs in the interface table external_ids interface for this interface
- **list port-other-configs** - a list of the key/value pairs in the port table other_config column for this port
- **list interface-other-configs** - a list of the key/value pairs in the interface table other_config column for this interface

2.3.3 Examples of OVSDB Southbound MD-SAL API

Connect to an OVSDB Node

This example RESTCONF command adds an OVSDB node object to the OVSDB Southbound configuration data store and attempts to connect to the OVSDB host located at the IP address 10.11.12.1 on TCP port 6640.

```
POST http://<host>:8181/restconf/config/network-topology:network-topology/topology/
↪ovsdb:1/
Content-Type: application/json
{
  "node": [
     {
       "node-id": "ovsdb:HOST1",
       "connection-info": {
         "ovsdb:remote-ip": "10.11.12.1",
         "ovsdb:remote-port": 6640
       }
     }
  ]
}
```

Query the OVSDB Southbound Configuration MD-SAL

Following on from the previous example, if the OVSDB Southbound configuration MD-SAL is queried, the REST-CONF command and the resulting reply is similar to the following example.

```
GET http://<host>:8080/restconf/config/network-topology:network-topology/topology/
↪ovsdb:1/
Application/json data in the reply
{
  "topology": [
    {
      "topology-id": "ovsdb:1",
      "node": [
        {
          "node-id": "ovsdb:HOST1",
          "ovsdb:connection-info": {
            "remote-port": 6640,
            "remote-ip": "10.11.12.1"
          }
        }
      ]
    }
  ]
}
```

2.3.4 Reference Documentation

Openvswitch schema

2.4 OVSDB Hardware VTEP Developer Guide

2.4.1 Overview

TBD

2.4.2 OVSDB Hardware VTEP Architecture

TBD

CHAPTER 3

OVSDB Design Specifications

Starting from Flourine, OVSDB uses RST format Design Specification document for all new features. These specifications are perfect way to understand various OVSDB features.

Contents:

Table of Contents

3.1 Title of the feature

[link to gerrit patch]

Brief introduction of the feature.

3.1.1 Problem description

Detailed description of the problem being solved by this feature

Use Cases

Use cases addressed by this feature.

3.1.2 Proposed change

Details of the proposed change.

Pipeline changes

Any changes to pipeline must be captured explicitly in this section.

Yang changes

This should detail any changes to yang models.

Configuration impact

Any configuration parameters being added/deprecated for this feature? What will be defaults for these? How will it impact existing deployments?

Note that outright deletion/modification of existing configuration is not allowed due to backward compatibility. They can only be deprecated and deleted in later release(s).

Clustering considerations

This should capture how clustering will be supported. This can include but not limited to use of CDTCL, EOS, Cluster Singleton etc.

Other Infra considerations

This should capture impact from/to different infra components like MDSAL Datastore, karaf, AAA etc.

Security considerations

Document any security related issues impacted by this feature.

Scale and Performance Impact

What are the potential scale and performance impacts of this change? Does it help improve scale and performance or make it worse?

Targeted Release

What release is this feature targeted for?

Alternatives

Alternatives considered and why they were not selected.

3.1.3 Usage

How will end user use this feature? Primary focus here is how this feature will be used in an actual deployment.

This section will be primary input for Test and Documentation teams. Along with above this should also capture REST API and CLI.

Features to Install

ovsdb

Identify existing karaf feature to which this change applies and/or new karaf features being introduced. These can be user facing features which are added to integration/distribution or internal features to be used by other projects.

REST API

Sample JSONS/URIs. These will be an offshoot of yang changes. Capture these for User Guide, CSIT, etc.

CLI

Any CLI if being added.

3.1.4 Implementation

Assignee(s)

Who is implementing this feature? In case of multiple authors, designate a primary assignee and other contributors.

Primary assignee: <developer-a>

Other contributors: <developer-b> <developer-c>

Work Items

Break up work into individual items. This should be a checklist on Trello card for this feature. Give link to trello card or duplicate it.

3.1.5 Dependencies

Any dependencies being added/removed? Dependencies here refers to internal [other ODL projects] as well as external [OVS, karaf, JDK etc.] This should also capture specific versions if any of these dependencies. e.g. OVS version, Linux kernel version, JDK etc.

This should also capture impacts on existing project that depend on OVSDB. Following projects currently depend on OVSDB: * Netvirt * SFC * Genius

3.1.6 Testing

Capture details of testing that will need to be added.

Unit Tests

Integration Tests

CSIT

3.1.7 Documentation Impact

What is impact on documentation for this change? If documentation change is needed call out one of the <contributors> who will work with Project Documentation Lead to get the changes done.

Don't repeat details already discussed but do reference and call them out.

3.1.8 References

Add any useful references. Some examples:

- Links to Summit presentation, discussion etc.
- Links to mail list discussions
- Links to patches in other projects
- Links to external documentation

[1] OpenDaylight Documentation Guide

[2] https://specs.openstack.org/openstack/nova-specs/specs/kilo/template.html

---

Note: This template was derived from [2], and has been modified to support our project.
