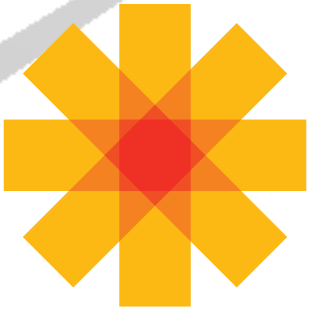
OPEN
DAYLIGHT

OpenDaylight

Getting Started Guide

Lithium (June 29, 2015)



OPEN
DAYLIGHT

OpenDaylight Getting Started Guide

OpenDaylight Community

Lithium (2015-06-29)

This guide describes how to get started with OpenDaylight.

Table of Contents

List of Tables

OpenDaylight Overview

The OpenDaylight project is a collaborative open source project that aims to accelerate adoption of Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) with a transparent approach that fosters new innovation.

OpenDaylight mainly consists of software designed to be run on top of a Java Virtual Machine (JVM) and can be run on any operating system and hardware as there is a Java Runtime Environment (JRE) available for it.

For a more detailed information about OpenDaylight, see the and *OpenDaylight User Guie*, *OpenDaylight Developer Guide*.

1. OpenDaylight Release Notes

Table of Contents

Target Environment

For Execution

The OpenDaylight Karaf container, OSGi bundles, and Java class files are portable and should run on any Java 7- or Java 8-compliant JVM to run. Certain projects and certain features of some projects may have additional requirements. Those are noted in the project-specific release notes.

Projects and features which have known additional requirements are: * TCP-MD5 requires 64-bit Linux * TSDR has extended requirements for external databases * Persistence has extended requirements for external databases * SFC requires addition features for certain configurations * SXP depends on TCP-MD5 on thus requires 64-bit Linux * SNBI has requirements for Linux and Docker * OpFlex requires Linux * DLUX requires a modern web browser to view the UI * AAA when using federation has additional requirements for external tools * VTN has components which require Linux

Note

If you are using the Oracle JDK, version 1.7.0_45 or later is required.

For Development

OpenDaylight is written primarily in Java project and primarily uses Maven as a build tool Consequently the two main requirements to develop projects within OpenDaylight are:

- A Java 7- or Java 8-compliant JDK
- Maven 3.1.1 or later

Applications and tools built on top of OpenDaylight using it's REST APIs should have no special requirements beyond whatever is needed to run the application or tool and make the REST calls.

In some places, OpenDaylight makes use of the Xtend language. While Maven will download the appropriate tools to build this, additional plugins may be required for IDE support.

The projects with additional requirements for execution typically have similar or more extensive additional requirements for development. See the project-specific release notes for details.

Known Issues and Limitations

Other than as noted in project-specific release notes, we know of the following limitations:

1. Migration from Helium to Lithium has not been extensively tested. The per-project release notes include migration and compatibility information when it is known. Broader support is anticipated in a later Lithium service release.

2. There are scales beyond which the controller has been unreliable when collecting flow statistics from OpenFlow switches. In tests, these issues became apparent when managing thousands of OpenFlow switches, however this may vary depending on deployment and use cases.

Project-specific Release Notes

For the release notes of individual projects, please see the following pages on the OpenDaylight Wiki.

- Authentication, Authorization and Accounting (AAA)
- ALTO
- BGP PCEP
- Controller
- Control And Provisioning of Wireless Access Points (CAPWAP)
- Device Identification and Driver Management (DIDM)
- DLUX
- Group Based Policy (GPB)
- Internet of Things Data Management (IoTDM)
- L2 Switch
- Link Aggregation Control Protocol (LACP)
- LISP Flow Mapping
- Network Intent Composition
- Neutron Northbound
- ODL Root Parent
- OpFlex
- OpenFlow Plugin
- OpenFlow Protocol Library

- OVSDB Integration
- Packet Cable/PCMM
- Persistence
- Reservation
- SDN Interface Application
- Secure Network Bootstrapping Infrastructure (SNBI)
- SNMP4SDN
- SNMP Plugin
- Secure tag eXchange Protocol (SXP)
- Service Function Chaining (SFC)
- TCP-MD5
- Time Series Data Repository (TSDR)
- Table Type Patterns (TTP)
- Topology Processing Framework
- Unified Secure Channel (USC)
- VPN Service
- Virtual Tenant Network (VTN)
- YANG Tools

Projects without Release Notes

The following projects participated in Lithium, but intentionally do not have release notes.

- **Documentation Project** produced this and the other downloadable documentation
- **Integration Group** hosted the OpenDaylight-wide tests and main release distribution
  - **Controller Core Functionality Tutorials** provided a single test suite (dsbenchmark) that was used as part of integration testing
- **Release Engineering - autorelease** was used to build the Lithium release artifacts and including the main release download.

2. Getting and Installing OpenDaylight Lithium

Table of Contents

Downloading and installing OpenDaylight Lithium

The default distribution can be found on the OpenDaylight software download page: http://www.opendaylight.org/software/downloads

The Karaf distribution has no features enabled by default. However, all of the features are available to be installed.

Note

For compatibility reasons, you cannot enable all the features simultaneously. We try to document known incompatibilities below.

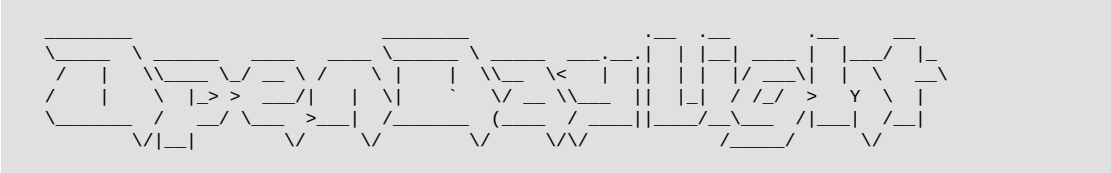
Running the karaf distribution

To run the Karaf distribution:

1. Unzip the zip file.
2. Navigate to the directory.
3. run `./bin/karaf`.

For Example:

```
$ ls distribution-karaf-0.3.0-Lithium.zip
distribution-karaf-0.3.0-Lithium.zip
$ unzip distribution-karaf-0.3.0-Lithium.zip
Archive:  distribution-karaf-0.3.0-Lithium.zip
   creating: distribution-karaf-0.3.0-Lithium/
   creating: distribution-karaf-0.3.0-Lithium/configuration/
   creating: distribution-karaf-0.3.0-Lithium/data/
   creating: distribution-karaf-0.3.0-Lithium/data/tmp/
   creating: distribution-karaf-0.3.0-Lithium/deploy/
   creating: distribution-karaf-0.3.0-Lithium/etc/
   creating: distribution-karaf-0.3.0-Lithium/externalapps/
...
  inflating: distribution-karaf-0.3.0-Lithium/bin/start.bat
  inflating: distribution-karaf-0.3.0-Lithium/bin/status.bat
  inflating: distribution-karaf-0.3.0-Lithium/bin/stop.bat
$ cd distribution-karaf-0.3.0-Lithium
$ ./bin/karaf
```



- Press **tab** for a list of available commands
- Typing **[cmd] –help** will show help for a specific command.
- Press **ctrl-d** or type **system:shutdown** or **logout** to shutdown OpenDaylight.

Installing the components

The section describes a list of components in OpenDaylight Lithium and the relevant Karaf feature to install in order to enable that component.

To install a feature use the following command:

```
feature:install
```

For Example:

```
feature:install <feature-name>
```

Multiple features can be installed using the following command:

```
feature:install <feature1-name> <feature2-name> ... <featureN-name>
```

Table 2.1. Lithium Components

Component Name	Component Description	Karaf feature name	Compatibility
ALTO	Enable support for Application-Layer Traffic Optimization	odl-alto-all	self+all
BGP	Enables support for BGP	odl-bgpcep-bgp-all	all
CAPWAP	Enables control of supported wireless APs	odl-capwap-ac-rest	all
DIDM	Device Identification and Driver Management	odl-didm-identification-api, odl-didm-identification, and odl-didm-drivers-api	all
Group Based Policy	Enable Endpoint Registry and Policy Repository REST APIs and associated functionality for Group Based Policy	odl-groupbasedpolicy-ofoverlay	self+all
Internet of Things Data Management	Enables support for the oneM2M specification	odl-iotdm-onem2m	all
L2 Switch	Provides L2 (Ethernet) forwarding across connected OppenFlow switches and support for host tracking	odl-l2switch-switch-ui	self+all
LACP	Enable support for the Link Aggregation Control Protocol	odl-lacp-ui	self+all
LISP Flow Mapping	Enable LISP control plane services including the mapping system services REST API and LISP protocol SB plugin	odl-lispflowmapping-all	all
MD-SAL Clustering	Provides support for operating a cluster of OpenDaylight instances	odl-mdsal-clustering	special
NETCONF over SSH	Provides support to manage NETCONF-enabled devices over SSH	odl-netconf-connector-ssh	all
Network Intent Composition	Enable support for high-level network control via intents	odl-nic-core	self+all

Component Name	Component Description	Karaf feature name	Compatibility
OVS Management	Enables OVS management using OVSDB plugin and its associated OVSDB northbound APIs	odl-ovsdb-all	all
OVSDB OpenStack Neutron	OpenStack Network Virtualization using OpenDaylight's OVSDB support	odl-ovsdb-openstack	self+all
OpFlex	Enables support for the OpFlex protocol	special (see user/ developer guide)	all
OpenFlow Flow Programming	Enables discovery and control of OpenFlow switches and the topology between them	odl-openflowplugin-flow-services-ui	all
OpenFlow Table Type Patterns	Allows OpenFlow Table Type Patterns to be manually associated with network elements	odl-ttp-all	all
PCEP	Enables support for PCEP	odl-bgpcep-pcep-all	all
Packetcable PCMM	Enables flow-based dynamic QoS management of CMTS using in the DOCSIS infrastructure	odl-packetcable-all	self+all
Packetcable Policy Server	Enables support for the PacketCable policy server	odl-packetcable-policy-server-all	self+all
RESTCONF API Support	Enables REST API access to the MD-SAL including the data store	odl-restconf	all
SDN Interface	Provides support for interaction and sharing of state between (non-clustered) OpenDaylight instances	odl-sdninterfaceapp-all	all
SFC over L2	Supports implementing SFC using Layer 2 forwarding	odl-sfcofl2	self+all
SFC over LISP	Supports implementing SFC using LISP	odl-sfclisp	all
SFC over REST	Supports implementing SFC using REST CRUD operations on network elements	odl-sfc-sb-rest	all
SFC over VXLAN	Supports implementing SFC using VXLAN tunnels	odl-sfc-ovs	self+all
SNMP Plugin	Enables monitoring and control of network elements via SNMP	odl-snmp-plugin	all
SNMP4SDN	Enables OpenFlow-like control of network elements via SNMP	odl-snmp4sdn-all	all
SSSD Federated Authentication	Enable support for federated authentication using SSSD	odl-aaa-sssd-plugin	all
Secure Networking Bootstrap	Defines a SNBI domain and associated white lists of devices to be accommodated to the domain	odl-snbi-all	self+all
Secure tag eXchange Protocol (SXP)	Enables distribution of shared tags to network devices	odl-sxp-controller	all
Service Flow Chaining (SFC)	Enables support for applying chains of network services to certain traffic	odl-sfc-all	all
Time Series Data Repository (TSDR)	Enables historical tracking of OpenFlow statistics	odl-tsdr-all	self+all
Topology Processing Framework	Enables merged and filtered views of network topologies	odl-topoprocessing-framework	all
Unified Secure Channel (USC)	Enables support for secure, remote connections to network devices	odl-usc-channel-ui	all
VPN Service	Enables support for OpenStack VPNaaS	odl-vpnservice-core	all
VTN Manager	Enables Virtual Tenant Network support	odl-vtn-manager-rest	self+all
VTN Manager Neutron	Enables OpenStack Neutron support of VTN Manager	odl-vtn-manager-neutron	self+all

In the table a compatibility value of **all** means that it can be run with other features. A value of **self+all** indicates that the feature can be installed with other features with a value of **all**, but may interact badly other features with a value of **self+all**.

Table 2.2. Experimental Lithium Components

Component Name	Component Description	Karaf feature name	Compatibility
Persistence	Enables saving of data to external databases	odl-persistence-api	self+all
Reservation	Enables bandwidth calendaring using the TL1 protocol	odl-reservation-models	all

Listing available features

To find the complete list of Karaf features, run the following command:

```
feature:list
```

To list the installed Karaf features, run the following command:

```
feature:list -i
```

Installing support for REST APIs

Most components that offer REST APIs will automatically load the RESTCONF API Support component, but if for whatever reason they seem to be missing, you can activate this support by installing the `odl-restconf` feature.

Installing the DLUX web interface

The OpenDaylight web interface; DLUX, draws information from topology and host databases to display information about the topology of the network, flow statistics, host locations. You can either use DLUX as a stand-alone plug-in or integrate with OpenDaylight. To install DLUX as a standalone application, refer to https://wiki.opendaylight.org/view/OpenDaylight_DLUX:Setup_and_Run To integrate with OpenDaylight you must enable DLUX Karaf feature. You can enable AD-SAL, MD-SAL and various other bundles within Karaf depending on the features you would like to access using DLUX. Each feature can be enabled or disabled separately.

Ensure that you have created a topology and enabled MD-SAL feature in the Karaf distribution before you use DLUX for network management. For more information about enabling the Karaf features for DLUX, refer to https://wiki.opendaylight.org/view/OpenDaylight_DLUX:DLUX_Karaf_Feature

Installing MD-SAL clustering

The MD-SAL clustering feature has "special" compatibility criteria. You **must** install clustering, before other features are installed. To install clustering, run the following command on the Karaf CLI console:

```
feature:install odl-mdsal-clustering
```

3. OpFlex agent-ovs Install Guide

Table of Contents

Required Packages

You'll need to install the following packages and their dependencies:

- libuv
- openvswitch-gbp
- openvswitch-gbp-lib
- openvswitch-gbp-kmod
- libopflex
- libmodelgbp
- agent-ovs

Packages are available for Red Hat Enterprise Linux 7 and Ubuntu 14.04 LTS. Some of the examples below are specific to RHEL7 but you can run the equivalent commands for upstart instead of systemd.

Note that many of these steps may be performed automatically if you're deploying this along with a larger orchestration system.

Host Networking Configuration

You'll need to set up your VM host uplink interface. You should ensure that the MTU of the underlying network is sufficient to handle tunneled traffic. We will use an example of setting up **eth0** as your uplink interface with a vlan of 4093 used for the networking control infrastructure and tunnel data plane.

We just need to set the MTU and disable IPv4 and IPv6 autoconfiguration. The MTU needs to be large enough to allow both the VXLAN header and VLAN tags to pass through without fragmenting for best performance. We'll use 1600 bytes which should be sufficient assuming you are using a default 1500 byte MTU on your virtual machine traffic. If you already have any NetworkManager connections configured for your uplink interface find the connection name and proceed to the next step. Otherwise, create a connection with (be sure to update the variable UPLINK_IFACE as needed):

```
UPLINK_IFACE=eth0
```

```
nmcli c add type ethernet ifname $UPLINK_IFACE
```

Now, configure your interface as follows:

```
CONNECTION_NAME="ethernet-$UPLINK_IFACE"
nmcli connection mod "$CONNECTION_NAME" connection.autoconnect yes \
    ipv4.method link-local \
    ipv6.method ignore \
    802-3-ethernet.mtu 9000 \
    ipv4.routes '224.0.0.0/4 0.0.0.0 2000'
```

Then bring up the interface with

```
nmcli connection up "$CONNECTION_NAME"
```

Next, create the infrastructure interface using the infrastructure VLAN (4093 by default). We'll need to create a vlan subinterface of your uplink interface, the configure DHCP on that interface. Run the following commands. Be sure to replace the variable values if needed. If you're not using NIC teaming, replace the variable team0 below

```
UPLINK_IFACE=team0
INFRA_VLAN=4093
nmcli connection add type vlan ifname $UPLINK_IFACE.$INFRA_VLAN dev
 $UPLINK_IFACE id $INFRA_VLAN
nmcli connection mod vlan-$UPLINK_IFACE.$INFRA_VLAN \
    ethernet.mtu 1600 ipv4.routes '224.0.0.0/4 0.0.0.0 1000'
sed "s/CLIENT_ID/01:$(ip link show $UPLINK_IFACE | awk '/ether/ {print $2}')/"
 \
    > /etc/dhcp/dhclient-$UPLINK_IFACE.$INFRA_VLAN.conf <<EOF
send dhcp-client-identifier CLIENT_ID;
request subnet-mask, domain-name, domain-name-servers, host-name;
EOF
```

Now bring up the new interface with:

```
nmcli connection up vlan-$UPLINK_IFACE.$INFRA_VLAN
```

If you were successful, you should be able to see an IP address when you run:

```
ip addr show dev $UPLINK_IFACE.$INFRA_VLAN
```

OVS Bridge Configuration

We'll need to configure an OVS bridge which will handle the traffic for any virtual machines or containers that are hosted on the VM host. First, enable the openvswitch service and start it:

```
# systemctl enable openvswitch
ln -s '/usr/lib/systemd/system/openvswitch.service' '/etc/systemd/system/
multi-user.target.wants/openvswitch.service'
# systemctl start openvswitch
# systemctl status openvswitch
openvswitch.service - Open vSwitch
   Loaded: loaded (/usr/lib/systemd/system/openvswitch.service; enabled)
   Active: active (exited) since Fri 2014-12-12 17:20:13 PST; 3s ago
  Process: 3053 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
 Main PID: 3053 (code=exited, status=0/SUCCESS)
Dec 12 17:20:13 ovs-server.cisco.com systemd[1]: Started Open vSwitch.
```

Next, we can create an OVS bridge (you may wish to use a different bridge name):

```
# ovs-vsctl add-br br0
# ovs-vsctl show
34aa83d7-b918-4e49-bcec-1b521acd1962
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
    ovs_version: "2.3.90"
```

Next, we configure a tunnel interface on our new bridge as follows:

```
# ovs-vsctl add-port br0 br0_vxlan0 -- \
    set Interface br0_vxlan0 type=vxlan \
    options:remote_ip=flow options:key=flow options:dst_port=8472
# ovs-vsctl show
34aa83d7-b918-4e49-bcec-1b521acd1962
    Bridge "br0"
        Port "br0_vxlan0"
            Interface "br0_vxlan0"
                type: vxlan
                options: {dst_port="8472", key=flow, remote_ip=flow}
        Port "br0"
            Interface "br0"
                type: internal
    ovs_version: "2.3.90"
```

Open vSwitch is now configured and ready.

Agent Configuration

Before enabling the agent, we'll need to edit its configuration file, which is located at "/etc/opflex-agent-ovs/opflex-agent-ovs.conf".

First, we'll configure the Opflex protocol parameters. If you're using an ACI fabric, you'll need the OpFlex domain from the ACI configuration, which is the name of the VMM domain you mapped to the interface for this hypervisor. Set the "domain" field to this value. Next, set the "name" field to a hostname or other unique identifier for the VM host. Finally, set the "peers" list to contain the fixed static anycast peer address of 10.0.0.30 and port 8009. Here is an example of a completed section (bold text shows areas you'll need to modify):

```
"opflex": {
    // The globally unique policy domain for this agent.
    "domain": "[CHANGE ME]",

    // The unique name in the policy domain for this agent.
    "name": "[CHANGE ME]",

    // a list of peers to connect to, by hostname and port.  One
    // peer, or an anycast pseudo-peer, is sufficient to bootstrap
    // the connection without needing an exhaustive list of all
    // peers.
    "peers": [
        {"hostname": "10.0.0.30", "port": 8009}
    ],
```

```
    "ssl": {
        // SSL mode.  Possible values:
        // disabled: communicate without encryption
        // encrypted: encrypt but do not verify peers
        // secure: encrypt and verify peer certificates
        "mode": "encrypted",

        // The path to a directory containing trusted certificate
        // authority public certificates, or a file containing a
        // specific CA certificate.
        "ca-store": "/etc/ssl/certs/"
    }
},
```

Next, configure the appropriate policy renderer for the ACI fabric. You'll want to use a stitched-mode renderer. You'll need to configure the bridge name and the uplink interface name. The remote anycast IP address will need to be obtained from the ACI configuration console, but unless the configuration is unusual, it will be 10.0.0.32.

```
// Renderers enforce policy obtained via OpFlex.
"renderers": {
    // Stitched-mode renderer for interoperating with a
    // hardware fabric such as ACI
    "stitched-mode": {
        "ovs-bridge-name": "br0",

        // Set encapsulation type.  Must set either vxlan or vlan.
        "encap": {
            // Encapsulate traffic with VXLAN.
            "vxlan" : {
                // The name of the tunnel interface in OVS
                "encap-iface": "br0_vxlan0",

                // The name of the interface whose IP should be used
                // as the source IP in encapsulated traffic.
                "uplink-iface": "eth0.4093",

                // The vlan tag, if any, used on the uplink interface.
                // Set to zero or omit if the uplink is untagged.
                "uplink-vlan": 4093,

                // The IP address used for the destination IP in
                // the encapsulated traffic.  This should be an
                // anycast IP address understood by the upstream
                // stitched-mode fabric.
                "remote-ip": "10.0.0.32"
            }
        },
        // Configure forwarding policy
        "forwarding": {
            // Configure the virtual distributed router
            "virtual-router": {
                // Enable virtual distributed router.  Set to true
                // to enable or false to disable.  Default true.
                "enabled": true,

                // Override MAC address for virtual router.
                // Default is "00:22:bd:f8:19:ff"
                "mac": "00:22:bd:f8:19:ff",
```

```
                // Configure IPv6-related settings for the virtual
                // router
                "ipv6" : {
                    // Send router advertisement messages in
                    // response to router solicitation requests as
                    // well as unsolicited advertisements.
                    "router-advertisement": true
                }
            },

            // Configure virtual distributed DHCP server
            "virtual-dhcp": {
                // Enable virtual distributed DHCP server.  Set to
                // true to enable or false to disable.  Default
                // true.
                "enabled": true,

                // Override MAC address for virtual dhcp server.
                // Default is "00:22:bd:f8:19:ff"
                "mac": "00:22:bd:f8:19:ff"
            }
        },

        // Location to store cached IDs for managing flow state
        "flowid-cache-dir": "DEFAULT_FLOWID_CACHE_DIR"
    }
}
```

Finally, enable the agent service:

```
# systemctl enable agent-ovs
ln -s '/usr/lib/systemd/system/agent-ovs.service' '/etc/systemd/system/multi-
user.target.wants/agent-ovs.service'
# systemctl start agent-ovs
# systemctl status agent-ovs
agent-ovs.service - Opflex OVS Agent
   Loaded: loaded (/usr/lib/systemd/system/agent-ovs.service; enabled)
   Active: active (running) since Mon 2014-12-15 10:03:42 PST; 5min ago
 Main PID: 6062 (agent_ovs)
   CGroup: /system.slice/agent-ovs.service
           ##6062 /usr/bin/agent_ovs
```

The agent is now running and ready to enforce policy. You can add endpoints to the local VM hosts using the OpFlex Group-based policy plugin from OpenStack, or manually.

4. OVSDB OpenStack Installation Guide

Table of Contents

Overview

This guide is geared towards installing OpenDaylight to use the OVSDB project to provide Neutron support for OpenStack.

Open vSwitch (OVS) is generally accepted as the unofficial standard for Virtual Switching in the Open hypervisor based solutions. For information on OVS, see Open vSwitch.

With OpenStack within the SDN context, 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/

Pre Requisites for Installing OVSDB OpenStack

- JRE 1.7+
- A distribution of OpenDaylight

Preparing for Installation

1. Download a copy of the latest OpenDaylight release in a Pre-Build archive of your choosing. Download OpenDaylight

Installing OVSDB OpenStack

1. Extract the OpenDaylight distribution somewhere.
2. Navigate to the /bin/ directory
3. Execute the karaf binary file, that should bring up the OpenDaylight console
4. Install the required features with these commands: feature:install odl-ovsdb-openstack

Sample output from the Karaf console

```
opendaylight-user@root>feature:list -i | grep ovsdb
```

```
odl-ovsdb-southbound-api             | 1.1.0-SNAPSHOT   | x         | odl-
ovsdb-southbound-1.1.0-SNAPSHOT
OpenDaylight :: southbound :: api
odl-ovsdb-southbound-impl            | 1.1.0-SNAPSHOT   | x         | odl-
ovsdb-southbound-1.1.0-SNAPSHOT
OpenDaylight :: southbound :: impl
odl-ovsdb-southbound-impl-rest       | 1.1.0-SNAPSHOT   | x         | odl-
ovsdb-southbound-1.1.0-SNAPSHOT
OpenDaylight :: southbound :: impl :: REST
odl-ovsdb-southbound-impl-ui         | 1.1.0-SNAPSHOT   | x         | odl-
ovsdb-southbound-1.1.0-SNAPSHOT
OpenDaylight :: southbound :: impl :: UI
odl-ovsdb-openstack                  | 1.1.0-SNAPSHOT   | x         | ovsdb-1.
1.0-SNAPSHOT
OpenDaylight :: OVSDB :: OpenStack Network Virtual
```

Verifying your Installation

To verify that the installation was successful, use the log:display command in karaf and check that there are no errors logs relating to odl-ovsdb-openstack.

Troubleshooting

There is no easy way to troubleshoot an installation of odl-ovsdb-openstack. Perhaps a combination of log:display | grep -i ovsdb in karaf, Open vSwitch commands (ovs-vsctl) and OpenStack logs will be useful but will not explain everything.

Uninstalling OVSDB OpenStack

1. Shutdown the karaf instance: system:shutdown
2. Remove what is in the /data folder of the OpenDaylight Distribution.

5. TSDR H2 Default Datastore Installation Guide

Table of Contents

This document is for the user to install the artifacts that are needed for using Time Series Data Repository (TSDR) functionality in the ODL Controller by enabling the default JPA (H2) Datastore. TSDR is new functionality added in OpenDaylight in Lithium Release.

Overview

In Lithium Release the time deries data records of OpenFlow statistics are collected periodically and stored in a persistent store. For non-production usage, the bundled default JPA based datastore (H2) is utilized based on odl-tsdr-all feature installation. The TSDR records get persisted in H2 store in <install folder>/tsdr/ folder by default.

Pre Requisites for Installing TSDR with default H2 datastore

There are no additional pre-requisites for TSDR based on default datastore

Preparing for Installation

No additional steps required for preparation of installing TSDR feature

Installing TSDR with default H2 datastore

Once OpenDaylight distribution is up, from karaf console install the TSDR feature with default datastore (JPA based datastore H2 store used) can be installed by

feature:install odl-tsdr-all

This will install all dependency features (and can take sometime) before returning control to the console.

Verifying your Installation

If the feature install was successful you should be able to see the following tsdr commands added

tsdr:list tsdr:purgeAll

Troubleshooting

Check the ../data/log/karaf.log for any exception related to TSDR or JPA related features

Post Installation Configuration

The feature installation takes care of automated configuration of the datasource by installing a file in <install folder>/etc named org.ops4j.datasource-metric.cfg. This contains the default location of <install folder>/tsdr where the H2 datastore files are stored. If you want to change the default location of the datastore files to some other location update the last portion of the url property in the org.ops4j.datasource-metric.cfg and then restart the karaf container

Upgrading From a Previous Release

Lithium being the first release supporting TSDR functionality, only fresh installation is possible.However if you want to move to production usage by enabling the store HBase for TSDR usage, you can do it by uninstalling the TSDR with default H2 datastore, restarting the Karaf container and then enabling the TSDR with HBase store as documented in tsdr-hbase-install.doc

Uninstalling TSDR with default H2 datastore

To uninstall the TSDR functionality with the default store, you need to do the following from karaf console * feature:uninstall odl-tsdr-all * feature:uninstall odl-tsdr-core * feature:uninstall odl-tsdr-H2-persistence

Its recommended to restart the Karaf container after uninstallation of the TSDR functionality with the default store

6. TSDR HBase Data Store Installation Guide

Table of Contents

This document is for the user to install the artifacts that are needed for using HBase Data Store in Time Series Data Repository, which is a new feature available in OpenDaylight Lithium release.

Overview

The Time Series Data Repository (TSDR) project in OpenDaylight (ODL) creates a framework for collecting, storing, querying, and maintaining time series data in the OpenDaylight SDN controller. It contains the following services and components:

- Data Collection Service
- Data Storage Service
- TSDR Persistence Layer with data stores as plugins
- TSDR Data Stores
- Data Query Service
- Data Aggregation Service
- Data Purging Service

Data Collection Service handles the collection of time series data into TSDR and hands it over to Data Storage Service. Data Storage Service stores the data into TSDR through TSDR Persistence Layer. TSDR Persistence Layer provides generic Service APIs allowing various data stores to be plugged in. Data Aggregation Service aggregates time series fine-grained raw data into course-grained roll-up data to control the size of the data. Data Purging Service periodically purges both fine-grained raw data and course-granined aggregated data according to user-defined schedules.

In Lithium, we implemented Data Collection Service, Data Storage Service, TSDR Persistence Layer, TSDR HBase Data Store, and TSDR H2 Data Store. Among these services and components, time series data is communicated using a common TSDR data model, which is designed and implemented for the abstraction of the time series data commonalities.

With these functions, TSDR will be able to collect the data from the data sources and store them into one of the TSDR data stores: either HBase Data Store or H2 Data Store. We also provided a simple query command from Karaf console for the user to retrieve TSDR data from the data stores.

A future release will contain Data Aggregation service, Data Purging Service, and a full-fledged Data Query Service with Norghbound APIs.

Prerequisites for Installing TSDR HBase Data Store

The hardware requirements are the same as those for standard ODL controller installation.

The software requirements for TSDR HBase Data Store are as follows:

- The supported operating system for TSDR HBase Data Store is Linux. We do not support TSDR HBase Data Store on Windows.
- Besides the software that ODL requires, we also require HBase database running on top of Hadoop single node.

Preparing for Installation

Download HBase (version number to be finalized) from the following website.

http://archive.apache.org/dist/hbase/hbase-0.94.15/

Installing TSDR HBase Data Store

Installing TSDR HBase Data Store contains two steps:

- Installing HBase server, and
- Installing TSDR HBase Data Store features from ODL Karaf console.

This installation guide will only cover the first step. For installing TSDR HBase Data Store features, please refer to TSDR HBase Data Store User Guide.

In Lithium, we only support HBase single node running together on the same machine as ODL controller. Therefore, follow the steps to download and install HBase server onto the same box as where ODL controller is running:

- Create a folder in Linux operating system for the HBase server.

For example, create an hbase directory under /usr/lib:

```
mkdir /usr/lib/hbase
```

- Unzip the downloaded HBase server tar file.

Run the following command to unzip the installation package:

```
tar xvf <hbase-installer-name>  /usr/lib/hbase
```

- Make proper changes in hbase-site.xml

  a. Under <hbase-install-directory>/conf/, there is a hbase-site.xml. Although it is not recommended, an experience user with HBase canmodify the data directory for hbase server to store the data.

  b. Modify the value of the property with name "hbase.rootdir" in the file to reflect the desired file directory for storing hbase data.

The following is an example of the file:

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>file:///usr/lib/hbase/data</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/usr/lib/hbase/zookeeper</value>
  </property>
</configuration>
```

Verifying your Installation

After the HBase server is properly installed, start hbase server and hbase shell.

a. start hbase server

```
cd <hbase-installation-directory>
./start-hbase.sh
```

b. start hbase shell

```
cd <hbase-insatllation-directory>
./hbase shell
```

Post Installation Configuration

Please refer to HBase Data Store User Guide.

Upgrading From a Previous Release

Lithium is the first release of TSDR. Upgrading is not applicable for TSDR Lithium release.

Uninstalling HBase Data Store

To uninstall TSDR HBase Data Store,

a. stop hbase server

```
cd <hbase-installation-directory>
./stop-hbase.sh
```

b. remove the file directory that contains the HBase server installation.

```
rm -r <hbase-installation-directory>
```

7. VTN Installation Guide

Table of Contents

Overview

OpenDaylight Virtual Tenant Network (VTN) is an application that provides multi-tenant virtual network on an SDN controller.

Conventionally, huge investment in the network systems and operating expenses are needed because the network is configured as a silo for each department and system. Therefore various network appliances must be installed for each tenant and those boxes cannot be shared with others. It is a heavy work to design, implement and operate the entire complex network.

The uniqueness of VTN is a logical abstraction plane. This enables the complete separation of logical plane from physical plane. Users can design and deploy any desired network without knowing the physical network topology or bandwidth restrictions.

VTN allows the users to define the network with a look and feel of conventional L2/L3 network. Once the network is designed on VTN, it will automatically be mapped into underlying physical network, and then configured on the individual switch leverage SDN control protocol. The definition of logical plane makes it possible not only to hide the complexity of the underlying network but also to better manage network resources. It achieves reducing reconfiguration time of network services and minimizing network configuration errors. OpenDaylight Virtual Tenant Network (VTN) is an application that provides multi-tenant virtual network on an SDN controller. It provides API for creating a common virtual network irrespective of the physical network.

It is implemented as two major components

- VTN Manager
- VTN Coordinator

VTN Manager

An OpenDaylight Controller Plugin that interacts with other modules to implement the components of the VTN model. It also provides a REST interface to configure VTN components in ODL controller. VTN Manager is implemented as one plugin to the OpenDaylight controller. This provides a REST interface to create/update/delete VTN

components. The user command in VTN Coordinator is translated as REST API to VTN Manager by the ODC Driver component. In addition to the above mentioned role, it also provides an implementation to the OpenStack L2 Network Functions API.

VTN Coordinator

The VTN Coordinator is an external application that provides a REST interface for a user to use the VTN Virtualization. It interacts with VTN Manager plugin to implement the user configuration. It is also capable of multiple controller orchestration. It realizes Virtual Tenant Network (VTN) provisioning in OpenDaylight Controllers (ODC). In the OpenDaylight architecture VTN Coordinator is part of the network application, orchestration and services layer. VTN Coordinator has been implemented as an external application to the OpenDaylight controller. This component is responsible for the VTN virtualization. VTN Coordinator will use the REST interface exposed by the VTN Manger to realize the virtual network using the OpenDaylight controller. It uses OpenDaylight APIs (REST) to construct the virtual network in ODCs. It provides REST APIs for northbound VTN applications and supports virtual networks spanning across multiple ODCs by coordinating across ODCs.

Preparing for Installation

VTN Manager

Running the Karaf distribution

Follow the instructions in Getting and Installing OpenDaylight.

VTN Coordinator

- Arrange a physical/virtual server with any one of the supported 64-bit OS environment.
  - RHEL 6 / 7
  - CentOS 6 / 7
- Install these packages

```
yum install perl-Digest-SHA uuid libxslt libcurl unixODBC json-c
```

```
rpm -ivh http://yum.postgresql.org/9.3/redhat/rhel-6-x86_64/pgdg-
redhat93-9.3-1.noarch.rpm
```

```
yum install postgresql93-libs postgresql93 postgresql93-server postgresql93-
contrib postgresql93-odbc
```

Installing VTN

VTN Manager

Install Feature

```
feature:install odl-vtn-manager-rest odl-vtn-manager-neutron
```

Note

The above command will install all features of VTN Manager. You can install only REST or Neutron also.

VTN Coordinator

- Enter into the externalapps directory in the top directory of Lithium

```
cd distribution-karaf-0.2.1-Lithium-SR1/externalapps
```

- Run the below command to extract VTN Coordinator from the tar.bz2 file in the externalapps directory.

```
tar –C/ -jxvf distribution.vtn-coordinator-6.0.0.1-Lithium-SR1-bin.tar.bz2
```

This will install VTN Coordinator to /usr/local/vtn directory. The name of the tar.bz2 file name varies depending on the version. Please give the same tar.bz2 file name which is there in your directory.

- Configuring database for VTN Coordinator

```
/usr/local/vtn/sbin/db_setup
```

- To start the Coordinator

```
/usr/local/vtn/bin/vtn_start
```

Using VTN REST API:

Get the version of VTN REST API using the below command, and make sure the setup is working.

```
curl --user admin:adminpass -H 'content-type: application/json' -X GET http://
<VTN_COORDINATOR_IP_ADDRESS>:8083/vtn-webapi/api_version.json
```

The response should be like this, but version might differ:

```
{"api_version":{"version":"V1.2"}}
```

Verifying your Installation

VTN Manager

- In the karaf prompt, type the below command to ensure that vtn packages are installed.

```
feature:list i | grep vtn
```

- Run any VTN Manager REST API

```
curl --user "admin":"admin" -H "Accept: application/json" -H \"Content-
type: application/json" -X GET \http://localhost:8282/controller/nb/v2/vtn/
default/vtns
```

VTN Coordinator

- ps –ef | grep unc will list all the vtn apps
- Run any REST API for VTN Coordinator version

Uninstalling VTN

VTN Manager

```
Feature:uninstall odl-vtnmanager-all
```

VTN Coordinator

```
/usr/local/vtn/bin/vtn_stop
```

```
Remove the usr/local/vtn folder
```