OpenDaylight as a Platform for Network Programmability
NANOG 72, February 19-21, 2018

Charles Eckel, Cisco DevNet
eckelcu@cisco.com
Agenda

• What is SDN
• What is OpenDaylight
• Network programmability
• Installation
• Example use cases
• Conclusions
What is SDN
Software Defined Networking (SDN)

- Control & Data Planes separation?
  - OpenFlow?
  - Logically centralized control Plane?
  - White label switches?

- This a valid & useful SDN use case, but...

- SDN can be defined more broadly:
  - Network is a source of vast amount of data...
  - ..that can be utilized by variety of SDN applications

- True power of SDN is network programmability
SDN - A Broader Definition

Generic feedback/control/policy loop between apps and the network
What Do We Need from an SDN Controller?

• A platform for deploying SDN applications
• Provide an SDN application development environment
  • Developer-friendly APIs to network elements (REST/JSON, pub/sub, etc.)
  • Network-level abstraction through topologies
  • Protocol independence for network-facing applications
What is OpenDaylight
The OpenDaylight Community

• Founded in February 2013
• Run by the Linux Foundation
• Eclipse Public License
• 15 founding companies provided software and developers
• 600+ contributors
• 2.5M+ lines of code
• Mostly Java

• First release “Hydrogen”
  • February 2014
• Release frequency
  • Roughly every 6 months
• Current release - “Nitrogen”
  • 7th release, Sept 26, 2017
  • SR1 released Nov 26, 2017
• Next release is Oxygen
  • March 2018
Software Architecture

- Java - enterprise-grade, cross-platform compatible language
- Java Interfaces - for event listening, specifications and forming patterns
- Maven – build system
- Karaf – based on OSGi, provides:
  - dynamic loading of bundles
  - registering dependencies and services exported
  - exchanging information across bundles
Network programmability
Why Network Programmability Matters

Network Expenses

- CAPEX: 33%
- OPEX: 67%

Source: Forrester

Deployment Speed

- Computing: 0 seconds
- Networking: 1000 seconds

Source: Open Compute Project
The Need for Something Better

- SNMP had failed
  - For configuration, that is
  - Extensive use in fault handling and monitoring
- CLI scripting
  - “Market share” 70%+

RFC 3535

Abstract

This document provides an overview of a workshop held by the Internet Architecture Board (IAB) on Network Management. The workshop was hosted by CNRI in Reston, VA, USA from June 4 thru June 6, 2002. The goal of the workshop was to continue the important dialog started between network operators and protocol developers, and to guide the IETFs focus on future work regarding network management.
Best Practices Coming Together

SNMP Experience

CLI Best Practices

Operations Requirements

NETCONF, RESTCONF and YANG
YANG
YANG
Data Modeling Language for Networking

- Modeling language, defined in RFC 6020
- Represents operational state, configuration, transactions, and notifications
- Defines semantics
  - Constraints (i.e. “MUSTs”)
  - Reusable structures
  - Built-in and derived types

In Summary:
YANG is a full, formal contract language with rich syntax and semantics for network data
Model Structure

- Data structured as a tree
- Main node types:
  - Container
  - List
  - Leaf List
  - Leaf
YANG Model Example

• Screenshot from network-topology.yang
• Container network-topology' with list of topology' items
• List items (leafs) have a 'name' which is also the key for the list
Tools to work with YANG Models

- pyang - An extensible YANG validator and converter in python
  - Source Code - [https://github.com/mbj4668/pyang](https://github.com/mbj4668/pyang)
  - Python Package - [https://pypi.python.org/pypi/pyang](https://pypi.python.org/pypi/pyang)
  - Command line tool

- YANG Explorer - YANG Browser and RPC Builder
  - [https://github.com/CiscoDevNet/yang-explorer](https://github.com/CiscoDevNet/yang-explorer)
  - Web Based GUI
  - More difficult to setup

- OpenDaylight Yang Tools – Tools supporting NETCONF and YANG, code generation from YANG models
  - [https://wiki.opendaylight.org/view/YANG_Tools:Main](https://wiki.opendaylight.org/view/YANG_Tools:Main)
Display a YANG Module

$ pyang -f tree <yang-file>
pyang Tip – JavaScript Tree Output

- Use `pyang -f jstree -p <model.yang> -o <output.html>`
- Produces collapsible Tree / HTML

```
```

<table>
<thead>
<tr>
<th>Element</th>
<th>Schema Type</th>
<th>Flags</th>
<th>Opt Status Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>network-topology</td>
<td>module</td>
<td></td>
<td>current /nt:network-topology</td>
</tr>
<tr>
<td>topology</td>
<td>container</td>
<td></td>
<td>current /nt:network-topology/nt:topology</td>
</tr>
<tr>
<td>topology-id</td>
<td>list</td>
<td></td>
<td>current /nt:network-topology/nt:topology/nt:topology-id</td>
</tr>
<tr>
<td>server-provided</td>
<td>leaf</td>
<td>config</td>
<td>current /nt:network-topology/nt:server-provided</td>
</tr>
<tr>
<td>topology-types</td>
<td>leaf</td>
<td>boolean</td>
<td>no config?</td>
</tr>
<tr>
<td>underlay-topology</td>
<td>list</td>
<td></td>
<td>current /nt:network-topology/nt:underlay-topology/nt:topology</td>
</tr>
<tr>
<td>node</td>
<td>list</td>
<td></td>
<td>current /nt:network-topology/nt:node</td>
</tr>
<tr>
<td>node-id</td>
<td>leaf</td>
<td>config</td>
<td>current /nt:network-topology/nt:node/nt:node-id</td>
</tr>
<tr>
<td>supporting-node</td>
<td>list</td>
<td></td>
<td>current /nt:network-topology/nt:node/nt:supporting-node/nt:node-id</td>
</tr>
<tr>
<td>termination-point</td>
<td>list</td>
<td></td>
<td>current /nt:network-topology/nt:node/nt:termination-point/nt:link</td>
</tr>
<tr>
<td>link</td>
<td>list</td>
<td></td>
<td>current /nt:network-topology/nt:link</td>
</tr>
<tr>
<td>link-id</td>
<td>leaf</td>
<td>config</td>
<td>current /nt:network-topology/nt:link/nt:link-id</td>
</tr>
<tr>
<td>source</td>
<td>container</td>
<td></td>
<td>current /nt:network-topology/nt:link/nt:source</td>
</tr>
<tr>
<td>destination</td>
<td>container</td>
<td></td>
<td>current /nt:network-topology/nt:link/nt:destination</td>
</tr>
<tr>
<td>supporting-link</td>
<td>list</td>
<td></td>
<td>current /nt:network-topology/nt:link/nt:supporting-link</td>
</tr>
</tbody>
</table>

© 2018 Cisco and/or its affiliates. All rights reserved.  Cisco Public
Building a Plugin/Application

1. **Yang Tools**
   - Yang Model
   - Generate APIs

2. **Maven Build Tools**
   - Generated API Definition
   - Create API Bundle

3. **Maven Build Tools**
   - Plugin source code
   - Create Plugin Bundle

4. **API” OSGI Bundle**
   - Deploy

5. **“Plugin” OSGI Bundle**
   - Deploy

Controller
NETCONF

IETF network management protocol

• Defined in RFC 4741 (2006), updated by RFC 6241 (2011)
• Distinguishes between configuration and state data
• Multiple configuration datastores (candidate, running, startup)
• Configuration change validation and transactions
• Selective data retrieval via filtering
• Streaming and playback of event notifications

In Summary:
NETCONF provides fundamental programming features for convenient and robust automation of network services
NETCONF Sessions

- NETCONF is connection-oriented
  - SSH, TLS as underlying transport
  - XML for payload
- NETCONF client establishes session with server
- Session establishment: <hello> exchange
  - Announce capabilities, modules, features
- Session termination
  - <close-session>, <kill-session>
NETCONF Commands

- get : to retrieve operational data
- get-config : to retrieve configuration data
- edit-config : to edit a device configuration
- copy-config : to copy a configuration to another data store (e.g. non-volatile memory)
- delete-config : to delete a configuration in a data store
RESTCONF

Restful API for YANG data models

- IETF RFC 8040
- Configuration data and operational state data exposed as resources
- Access data using REST verbs (GET / PUT / POST …)
- Construct URIs, based on structure of YANG model, to access data
- HTTP instead of SSH for transport
- JSON in addition to XML for data encoding

In Summary:
RESTCONF provides light weight interface to network datastores leveraging well known combination of REST and JSON
RESTCONF URI & JSON Example

http://localhost:8181/restconf/config/network-topology:
  network-topology/topology/topology-netconf/node/vpp1

<nodel xmlns="urn:TBD:params:xml:ns:yang:network-topology">
  <node-id>vpp1</node-id>
  <host xmlns="urn:opendaylight:netconf-node-topology">{{vpp1_address}}</host>
  <port xmlns="urn:opendaylight:netconf-node-topology">2831</port>
  <username xmlns="urn:opendaylight:netconf-node-topology">admin</username>
  <password xmlns="urn:opendaylight:netconf-node-topology">admin</password>
  <tcp-only xmlns="urn:opendaylight:netconf-node-topology">false</tcp-only>
  <keepalive-delay xmlns="urn:opendaylight:netconf-node-topology">0</keepalive-delay>
</node>
High Level Manageability Architecture

Application
- ANY (C, Java, Python)
- NETCONF client
- RESTCONF client

Transport
- YANG-based XML
- SSH / TLS
- YANG-based XML/XML/JSON
- HTTPS

Network Device
- Manageability Infra
  - Config DB
  - BGP
  - QoS
  - VXLAN

Manageability
- Infra
- Config
- DB

ANY (Java, Python, Perl, PHP)

ANY (Java, Python, Perl, PHP)
Mounting YANG Datastores
OpenDaylight NETCONF Node “Discovery”

- Nodes added by POSTing to config:modules
- OpenDaylight connects to each node
- OpenDaylight learns capabilities (YANG modules) and stores to model cache
  - Cache at ~/cache/schema. Filenames of form yang-model@2016-07-12.yang.
# Distributions

https://www.opendaylight.org/technical-community/getting-started-for-developers/downloads-and-documentation

## Downloads

<table>
<thead>
<tr>
<th>Release</th>
<th>Release date</th>
<th>Downloads</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carbon SR2</td>
<td>October 16, 2017</td>
<td>• Pre-Built Tar</td>
<td>• Getting Started Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Pre-Built Zip</td>
<td>• Developers Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• NeXT UI</td>
<td>• User Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Virtual Tenant Network (VTN) Coordinator</td>
<td>• Installation Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Using OpenDaylight with OpenStack</td>
</tr>
<tr>
<td>Nitrogen SR1</td>
<td>November 28, 2017</td>
<td>• Pre-Built Tar</td>
<td>• Getting Started Guide</td>
</tr>
<tr>
<td>(Current Release)</td>
<td></td>
<td>• Pre-Built Zip</td>
<td>• Developers Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Virtual Tenant Network (VTN) Coordinator</td>
<td>• User Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• OpFlex</td>
<td>• Installation Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Using OpenDaylight with OpenStack</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Release Notes</td>
</tr>
</tbody>
</table>
$ unzip karaf-0.7.1.zip
Archive: karaf-0.7.1.zip
    creating: karaf-0.7.1/system/ ...
$ cd karaf-0.7.1
$ ./bin/karaf
karaf: Enabling Java debug options: -Xdebug -Xnoagent -Djava.compiler=NONE
     -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005
Listening for transport dt_socket at address: 5005
Apache Karaf starting up. Press Enter to open the shell now...
100% [========================================================================]
Karaf started in 0s. Bundle stats: 10 active, 10 total

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
Install Features using Karaf

- OpenDaylight distro comes without any features enabled by default
- All features are available for you to install
  - `feature:list` list all features available
  - `feature:list -i` list all features installed
  - `feature:install <feature>` install the `<feature>` feature
  - `feature:install <feature-1> <feature-2> … <feature-n>` install list of features
  - `feature:uninstall <feature>` uninstalls the `<feature>` feature
Install DLUX, NETCONF, and RESTCONF

```bash
opendaylight_user@root> feature:install odl-dlux-core
opendaylight_user@root> feature:install odl-dluxapps-yangui
opendaylight_user@root> feature:install odl-restconf-all
opendaylight_user@root> feature:install odl-netconf-all
opendaylight_user@root> feature:install odl-netconf-topology
Opendaylight_user@root> feature:install odl-netconf-connector-ssh
opendaylight_user@root> feature:list -r
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Required</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>odl-netconf-topology</td>
<td>1.3.1</td>
<td>x</td>
<td>Started</td>
</tr>
<tr>
<td>odl-restconf-all</td>
<td>1.6.1</td>
<td>x</td>
<td>Started</td>
</tr>
<tr>
<td>odl-netconf-connector-ssh</td>
<td>1.3.1</td>
<td>x</td>
<td>Started</td>
</tr>
<tr>
<td>odl-dluxapps-yangui</td>
<td>0.6.1</td>
<td>x</td>
<td>Started</td>
</tr>
<tr>
<td>odl-netconf-all</td>
<td>1.3.1</td>
<td>x</td>
<td>Started</td>
</tr>
<tr>
<td>odl-dlux-core</td>
<td>0.6.1</td>
<td>x</td>
<td>Started</td>
</tr>
<tr>
<td>wrap</td>
<td>0.0.0</td>
<td>x</td>
<td>Started</td>
</tr>
<tr>
<td>standard</td>
<td>4.0.10</td>
<td>x</td>
<td>Started</td>
</tr>
</tbody>
</table>
http://localhost:8181/index.html#/yangui/index
Example Use Cases
Mininet, OVSDB and OpenFlow
Cisco IOS XR using BGP-LS and PCE-P

- Cisco XRv topology in dCloud
  - dCloud is [http://dcloud.cisco.com](http://dcloud.cisco.com) (requires CCO login)
  - “OpenDaylight Boron SR3 with Apps with 8 Nodes v1”
  - ODL runs in dCloud (or use anyconnect/openconnect VPN to use local ODL instance)

- Use Pathman-SR application to create Segment Routed LSPs
VPP/Honeycomb using NETCONF and RESTCONF

- VPP is a high-performance, open source, software forwarder
  - [http://www.fd.io](http://www.fd.io)

- Honeycomb provides NETCONF and RESTCONF interfaces to VPP
OpenDaylight with Mininet – Step by Step

- Install, setup, and start Mininet VM using VirtualBox
  - Login (user=mininet, password=mininet)

- Within OpenDaylight, enable required feature set
  - `opendaylight-user@root> feature:install odl-l2switch-switch odl-dlux-core odl-dluxapps-applications`

- Within Mininet VM, start 3 switches controlled by OpenDaylight
  - `mininet@mininet-vm:~$ sudo mn --topo linear,3 --mac --controller=remote,ip=<OpenDaylight-IP>,port=6633 --switch ovs,protocols=OpenFlow13`
  - `mininet@mininet-vm:~$ pingall`

- From browser, log into OpenDaylight DLUX
Mininet Network Start

```
[ Mininet@mininet-vm ]$ sudo mn --topo linear,3 --mac --controller=remote,ip=192.168.40.18, port=6633 --switch ovs,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
[ Mininet@mininet-vm ]$ pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
Mininet> 
```
Using DLUX

• From Browser, log into OpenDaylight DLUX
  • http://<OpenDaylight-IP>:8181/index.html
    (credentials: admin/admin)

• Check out the network and switches by clicking on Nodes, Node Connectors
REST APIs

• Click on Yang UI and Expand All to see the REST APIs available
Inventory of Network Nodes

- GET opendaylight-inventory -> operational -> nodes
VPP/Honeycomb using NETCONF and RESTCONF

Step by Step
1. Create VM for Honeycomb and VPP
2. Install VPP and Honeycomb on VM
3. Start VPP and Honeycomb
4. Connect to VPP using CLI
5. Add interface(s) to VPP
6. Connect to VPP using Honeycomb/NETCONF
7. Connect to VPP using OpenDaylight
VPP/Honeycomb using NETCONF and RESTCONF

1. Create VM for Honeycomb and VPP

   • Download minimal CentOS 7 from https://www.centos.org/download/

   • Create VM and enable ssh using http://www.jeramysingleton.com/install-centos-7-minimal-in-virtualbox/ to create VM and enable ssh
     • Add two host-only adapters with DHCP and promiscuous mode enabled
       • One for VPP, another to access Honeycomb directly from laptop
     • To add sudo for my user (devnet/devnet) using https://www.digitalocean.com/community/tutorials/how-to-create-a-sudo-user-on-centos-quickstart
VPP/Honeycomb using NETCONF and RESTCONF

2. Install VPP and Honeycomb on VM

- FD.io wiki provides instructions for **installing VPP** and **installing HC**
  - Add the FD.io repo:
    - Add the following lines to `/etc/yum.repos.d/honeycomb-release.repo`
      ```
      [honeycomb-release]
      name=honeycomb release branch latest merge
      baseurl=https://nexus.fd.io/content/repositories/fd.io.centos7/
      enabled=1
      gpgcheck=0
      ```
  - Install both packages
    - `sudo yum install vpp`
    - `sudo yum install honeycomb`
VPP/Honeycomb using NETCONF and RESTCONF

3. Start VPP and Honeycomb

• Reset iptables
  • sudo ./iptables-reset.sh

• Flush interface to be used for DPDK
  • sudo ifconfig enp0s8 down
  • sudo ip add flush dev enp0s8

• Start VPP, then Honeycomb
  • sudo service vpp start
  • sudo service honeycomb start

• Check availability of Honeycomb’s SSH/NETCONF port:
  • netstat -an | grep 2831
VPP/Honeycomb using NETCONF and RESTCONF

4. Connect to VPP Using CLI

- Connect to VPP’s command line interface (CLI)
  • $ ssh devnet@192.168.60.101
  • $ sudo vppctl

        _______ \ ( )\  \\
/ / / / / / / / / / \\
/ / / / / / / / / / / \\
/ / / / / / / / / / / \\
/ / / / / / / / / / / \\
\//\//\//\//\//\//\//\//\//

• $vpp# show interface
  Name           Idx  State
  GigabitEthernet0/8/0  1   down
  local0          0   down
VPP/Honeycomb using NETCONF and RESTCONF

5. Add interface(s) to VPP

• Add a virtual interface using https://wiki.fd.io/view/VPP/Progressive_VPP_Tutorial#Exercise:_Create_an_Interface

• Optionally add a physical NIC using https://wiki.fd.io/view/VPP/How_To_Connect_A_PCI_Interface_To_VPP
  • Need to have associated a host-only network; if none, add one with DHCP and promiscuous mode before proceeding, should get something like
  • Details in notes section of slide
6. Connect to VPP Using Honeycomb and NETCONF

- Honeycomb listens on port 2831 for SSH/NETCONF
- Connect to VPP and issue for sample commands using: https://wiki.fd.io/view/Honeycomb/Releases/1609/Running_Honeycomb
- You also need to add ssh-dss when connecting via ssh
  - $ ssh -oHostKeyAlgorithms=+ssh-dss admin@192.168.60.101 -p 2831 -s netconf
- By default, honeycomb listens for RESTCONF on localhost:2831. To connect via RESTCONF from off-box
  - $ sudo vi /opt/honeycomb/config/restconf.json
    - Change restconf config from localhost or 127.0.0.1 to 0.0.0.0, e.g.
      "restconf-binding-address": "0.0.0.0",
      "restconf-port": 8183,
VPP/Honeycomb using NETCONF and RESTCONF

7. Connect to VPP Using OpenDaylight

- Import Postman environment

- Import Postman collection

- Add VPP to OpenDaylight topology with Postman
  - PUT
    - [http://{odl_address}:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/vpp1](http://{odl_address}:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/vpp1)

- View configuration in OpenDaylight DLUX
### Add VPP1

**PUT**

```plaintext
http://[odl_address]:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/vpp1
```

**Authorization**
- Basic YWRtaW46YWRtaW4=

**Accept**
- application/xml

**Content-Type**
- application/xml

**Body**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Domain</th>
<th>Path</th>
<th>Expires</th>
<th>HTTP</th>
<th>Secure</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSESSIONID</td>
<td>1ap882gt7pwk1rgeo2pwm16</td>
<td>localhost</td>
<td>/restconf</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

**Cookies**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Domain</th>
<th>Path</th>
<th>Expires</th>
<th>HTTP</th>
<th>Secure</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSESSIONID</td>
<td>1ap882gt7pwk1rgeo2pwm16</td>
<td>localhost</td>
<td>/restconf</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Conclusions
Key Takeaways

- SDN is more than just OpenFlow
- Network programmability is key benefit of SDN
- OpenDaylight provides a platform for network applications and programmable network infrastructure
Thank you!