OpenDaylight as a Platform for Network Programmability

NANOG 72, February 19-21, 2018

Charles Eckel, Cisco DevNet
eckelcu@cisco.com

Agenda

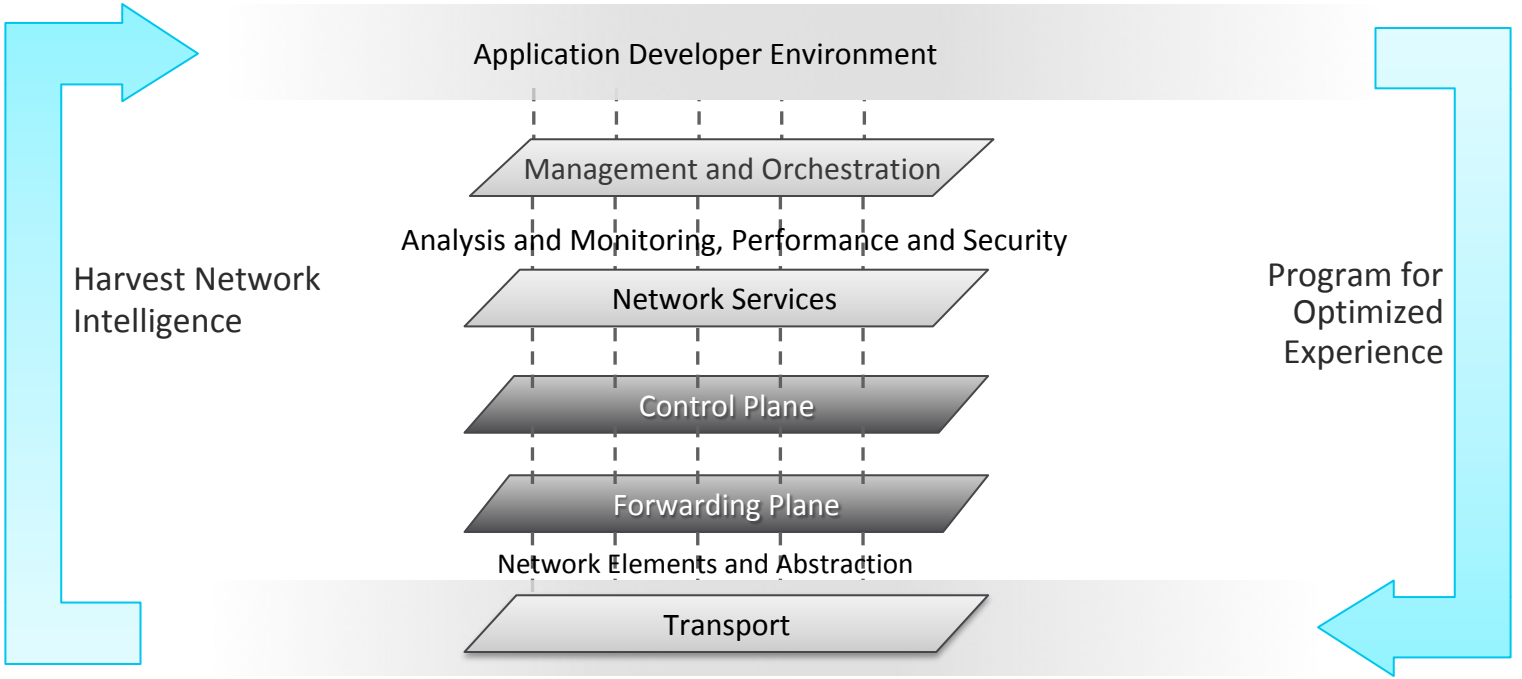
- What is SDN
- What is OpenDaylight
- Network programmability
- Installation
- Example use cases
- Conclusions

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

Graphical User Interface Application and Toolkit (DLUX / NeXT UI)

AAA AuthN Filter

OpenDaylight APIs REST/RESTCONF/NETCONF/AMQP

Northbound APIs to Orchestrators and Applications

Base Network Functions

- Host Tracker
- L2 Switch
- OpenFlow Forwarding Rules Mgr
- OpenFlow Stats Manager
- OpenFlow Switch Manager
- Topology Processing

Enhanced Network Services

- AAA
- Centinel – Streaming Data Hdlr
- Controller Shield
- Dev Discovery, ID & Drvr Mgmt
- DOCSIS Abstraction
- Link Aggregation Ctl Protocol
- LISP Service
- Messaging 4Transport
- NetIDE
- Neutron Northbound
- OVSDB Neutron
- SDN Integration Aggregator
- Service Function Chaining
- SNMP4SDN
- Time Series Data Repository
- Unified Secure Channel Mgr
- User Network Interface Mgr
- Virtual Private Network
- Virtual Tenant Network Mgr.

Network Abstractions

- ALTO Protocol Manager
- Fabric as a Service
- Group Based Policy Service
- NEMO
- Network Intent Composition

Controller Platform Services/Applications

Data Store (Config & Operational)

Service Abstraction Layer/Core

Messaging (Notifications / RPCs)

OpenFlow 1.0 1.3 TTP | OF-Config | OVSDB | NETCONF | LISP | BGP | PCEP | CAPWAP | OPFLEX | SXP | SNMP | USC | SNBI | IoT Http/CoAP | LACP | PCMM /COPS

Southbound Interfaces & Protocol Plugins

OpenFlow Enabled Devices

Open vSwitches

Additional Virtual & Physical Devices

Data Plane Elements (Virtual Switches, Physical Device Interfaces)

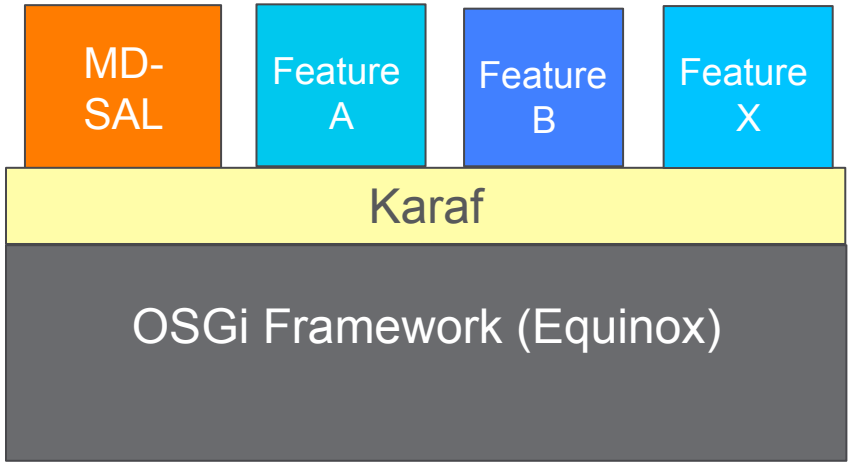
The OpenDaylight Community

- 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

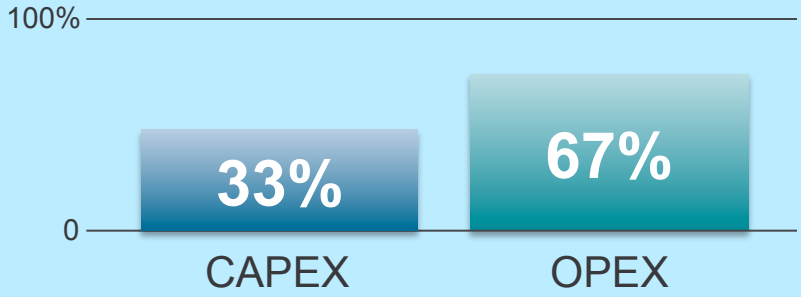
- Java - enterprise-grade, cross-platform compatible language
- Java Interfaces - for event listening, specifications and forming patterns
- Maven – build system
- Karaf – based on OSGi, provides:
  - dynamic loading of bundles
  - registering dependencies and services exported
  - exchanging information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



Network programmability

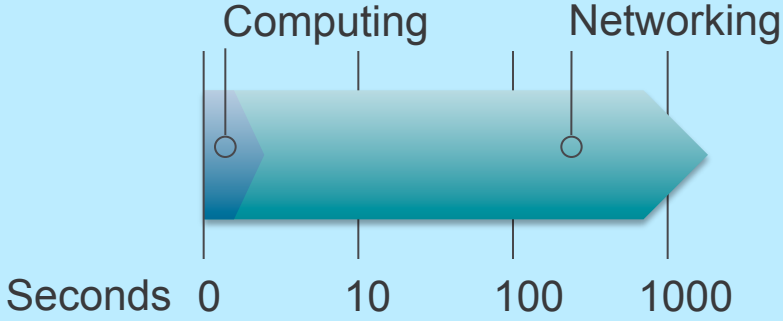
Why Network Programmability Matters

Network Expenses



Source: Forrester

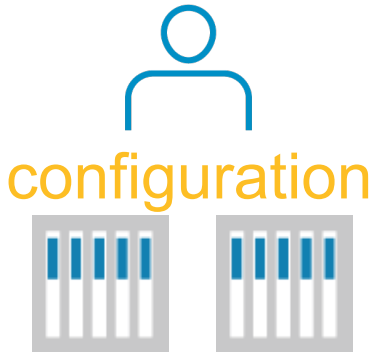
Deployment Speed



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%+

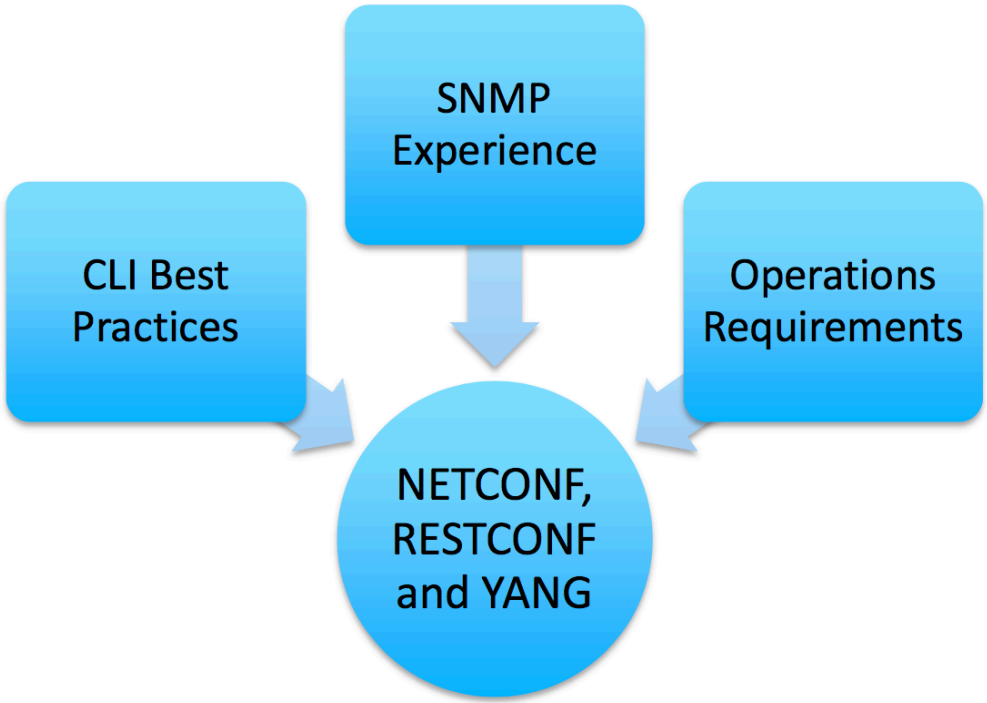


RFC 3535

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

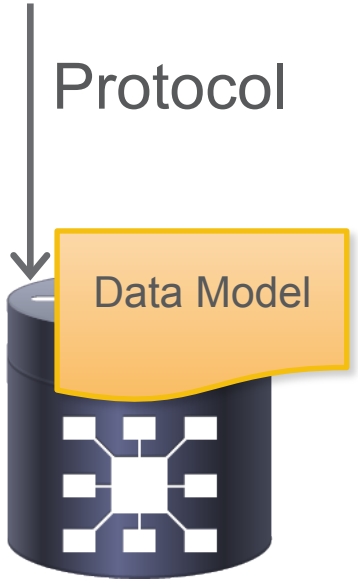


YANG

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

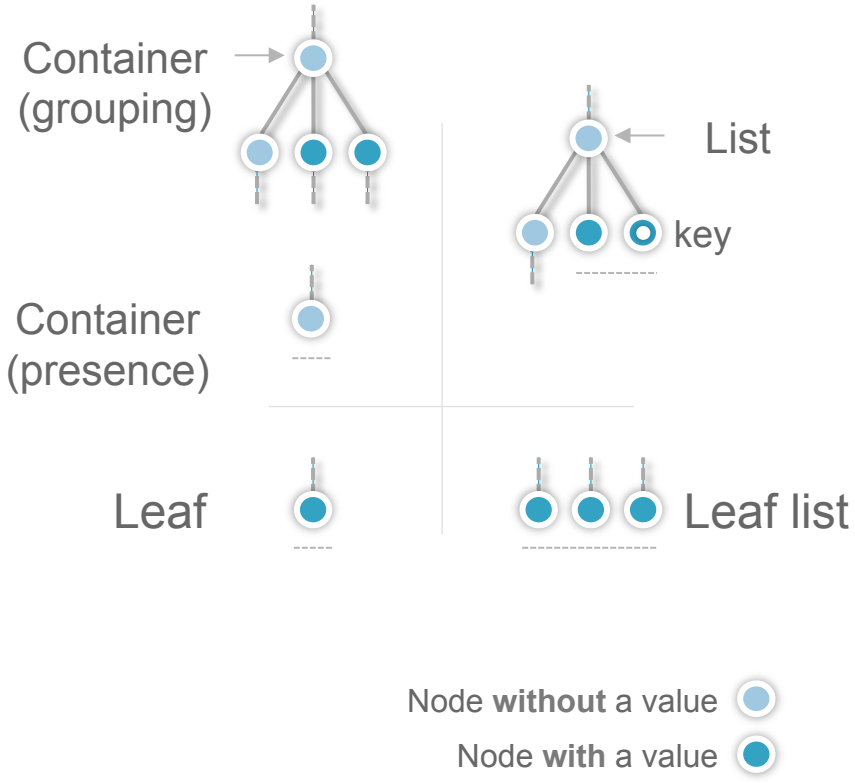


In Summary:

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

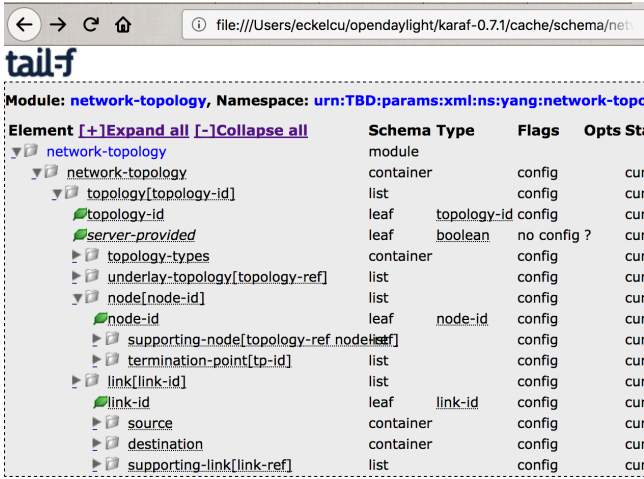
- Screenshot from network-topology.yang
- Container network-topology' with list of topology' items
- List items (leafs) have a 'name' which is also the key for the list

```
container network-topology {
    list topology {
        description "
            This is the model of an abstract topology.
            A topology contains nodes and links.
            Each topology MUST be identified by
            unique topology-id for reason that a network could contain many
            topologies.
        ";
        key "topology-id";
        leaf topology-id {
            type topology-id;
            description "
                It is presumed that a datastore will contain many topologies. To
                distinguish between topologies it is vital to have UNIQUE
                topology identifiers.
            ";
        }
        leaf server-provided {
            type boolean;
            config false;
            description "
                Indicates whether the topology is configurable by clients,
                or whether it is provided by the server.  This leaf is

                populated by the server implementing the model.
                It is set to false for topologies that are created by a client;
                it is set to true otherwise.  If it is set to true, any
                attempt to edit the topology MUST be rejected.
            ";
        }
        container topology-types {
            description
```

Tools to work with YANG Models

- pyang - An extensible YANG validator and converter in python
  - Source Code - https://github.com/mbj4668/pyang
  - Python Package - https://pypi.python.org/pypi/pyang
  - Command line tool
- YANG Explorer - YANG Browser and RPC Builder
  - 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

```
ECKELCU-M-H15L:schema eckelcu$ pyang -f tree network-topology
\@2013-10-21.yang
module: network-topology
   +--rw network-topology
      +--rw topology* [topology-id]
         +--rw topology-id              topology-id
         +--ro server-provided?         boolean
         +--rw topology-types
         +--rw underlay-topology* [topology-ref]
         |  +--rw topology-ref          topology-ref
         +--rw node* [node-id]
         |  +--rw node-id               node-id
         |  +--rw supporting-node* [topology-ref node-ref]
         |  |  +--rw topology-ref       topology-ref
         |  |  +--rw node-ref           node-ref
         |  +--rw termination-point* [tp-id]
         |     +--rw tp-id         tp-id
         |     +--ro tp-ref*       tp-ref
         +--rw link* [link-id]
            +--rw link-id               link-id
            +--rw source
            |  +--rw source-node       node-ref
            |  +--rw source-tp?        tp-ref
            +--rw destination
            |  +--rw dest-node         node-ref
            |  +--rw dest-tp?          tp-ref
            +--rw supporting-link* [link-ref]
               +--rw link-ref         link-ref
```



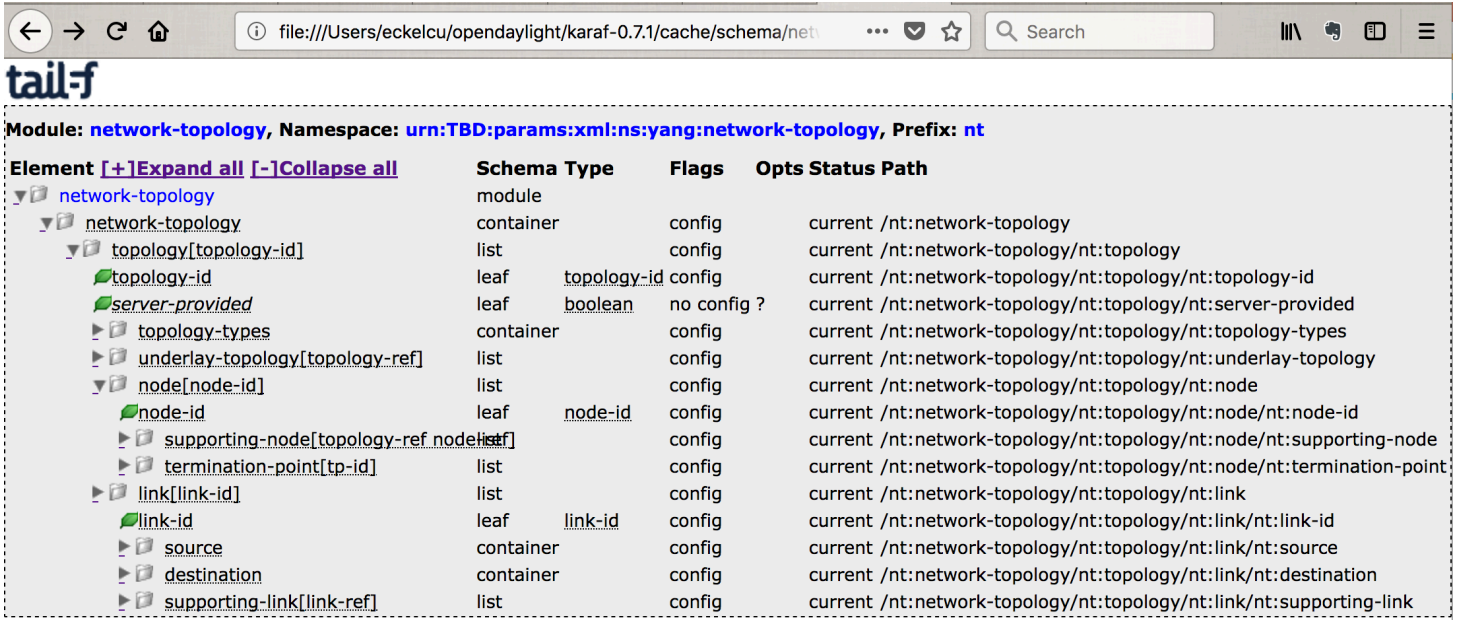
Display a YANG Module

```
$ pyang -f tree <yang-file>
```

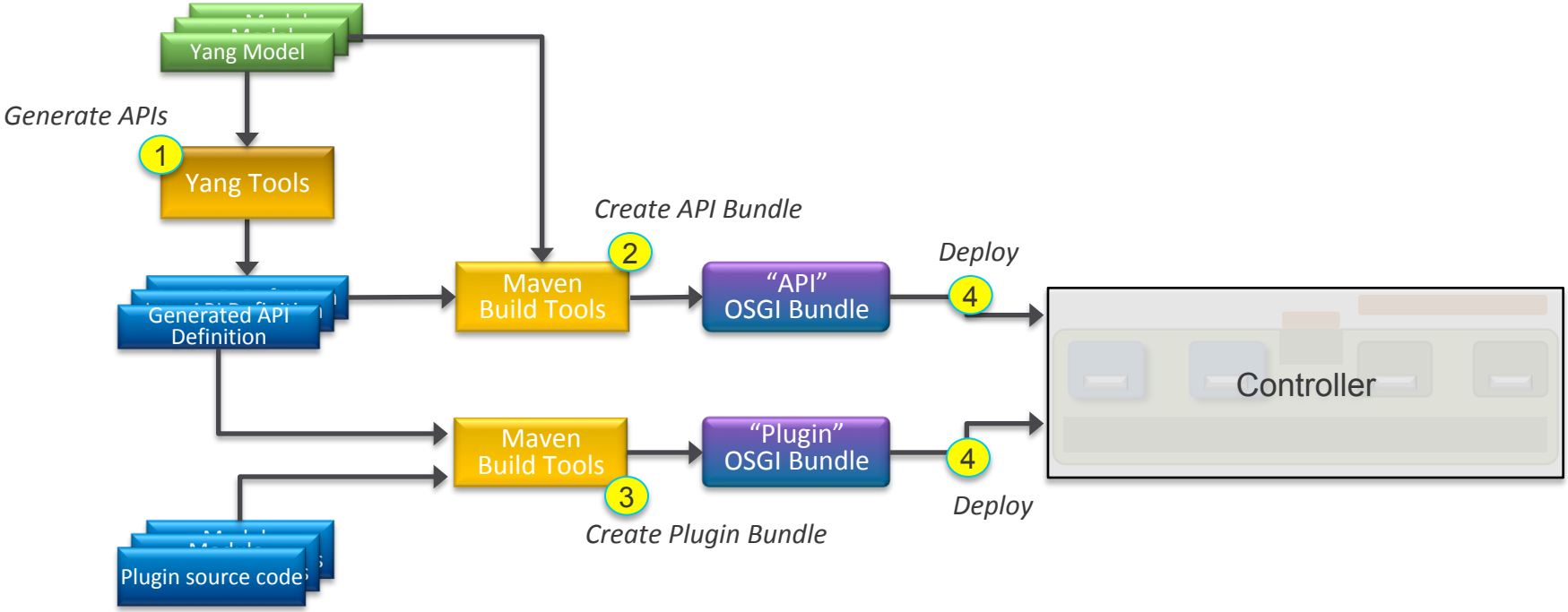
```
ECKELCU-M-H15L:schema eckelcu$ pyang -f tree network-topology
\@2013-10-21.yang
module: network-topology
   +--rw network-topology
      +--rw topology* [topology-id]
         +--rw topology-id          topology-id
         +--ro server-provided?     boolean
         +--rw topology-types
         +--rw underlay-topology* [topology-ref]
         |  +--rw topology-ref    topology-ref
         +--rw node* [node-id]
         |  +--rw node-id              node-id
         |  +--rw supporting-node* [topology-ref node-ref]
         |  |  +--rw topology-ref    topology-ref
         |  |  +--rw node-ref        node-ref
         |  +--rw termination-point* [tp-id]
         |     +--rw tp-id     tp-id
         |     +--ro tp-ref*   tp-ref
         +--rw link* [link-id]
            +--rw link-id            link-id
            +--rw source
            |  +--rw source-node    node-ref
            |  +--rw source-tp?     tp-ref
            +--rw destination
            |  +--rw dest-node    node-ref
            |  +--rw dest-tp?     tp-ref
            +--rw supporting-link* [link-ref]
               +--rw link-ref    link-ref
```

pyang Tip – JavaScript Tree Output

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



Building a Plugin/Application



NETCONF

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>

1. The NETCONF client establishes an SSH session to the NETCONF server.

2. The NETCONF client and server exchange NETCONF **hello** messages to exchange capabilities.

3. Now that the NETCONF client and server have exchanged **hello** messages, the client may issue an RPC. In this scenario, the client sends a **get** operation and the server responds with operational data. Note that the **get** operational should be filtered for specific data. Filters are built using XML.

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

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

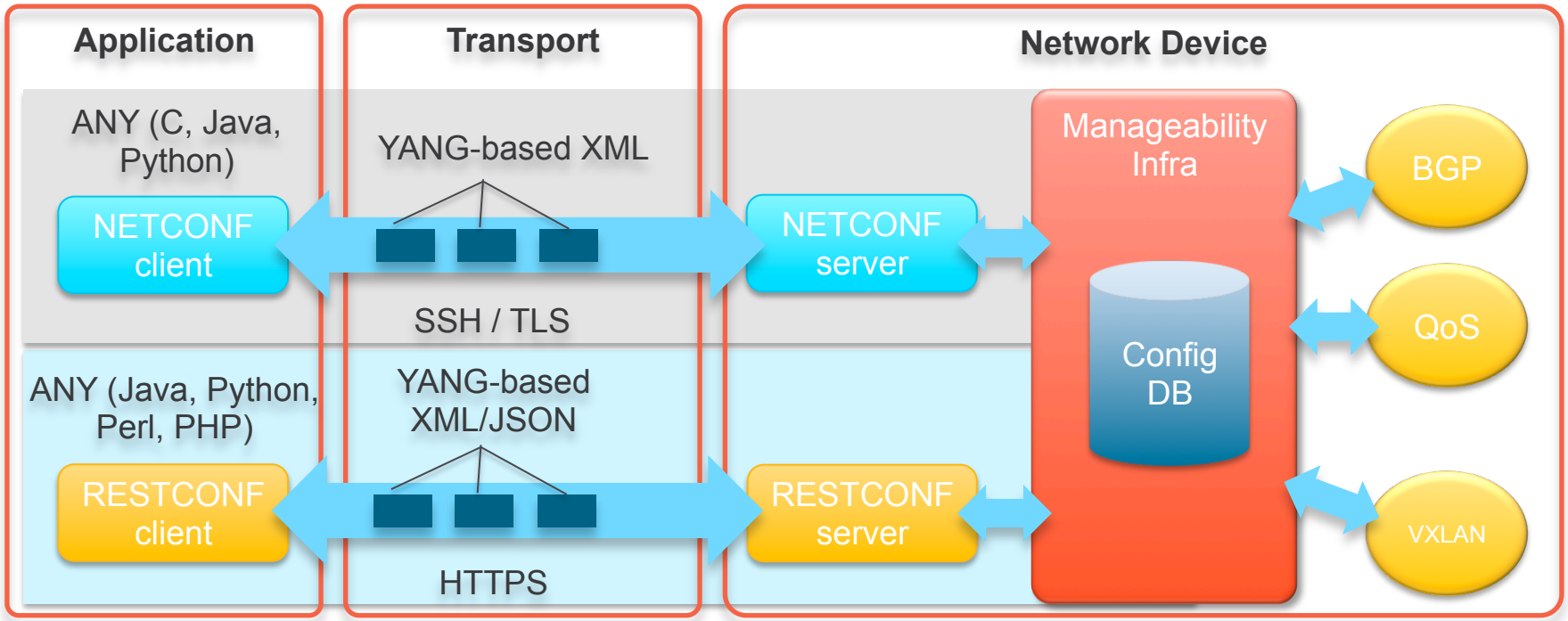
```
module: network-topology
   +--rw network-topology
      +--rw topology* [topology-id]
         +--rw topology-id                topo
         +--ro server-provided?           boo
         +--rw topology-types
         +--rw underlay-topology* [topo
         |  +--rw topology-ref    topol
         +--rw node* [node-id]
         |  +--rw node-id                 
         |  +--rw supporting-node* [top
         |  |  +--rw topology-ref     to
         |  |  +--rw node-ref         no
         |  +--rw termination-point* [t
```

```
module: netconf-node-topology
augment /nt:network-topology/nt:topolog
   +--rw topology-netconf
augment /nt:network-topology/nt:topolog
   +--rw (credentials)?
   |  +--:(login-password)
   |     +--rw username?
   |     +--rw password?
   +--rw host?
   +--rw port?
   +--rw tcp-only?
   +--rw schemaless?
```

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

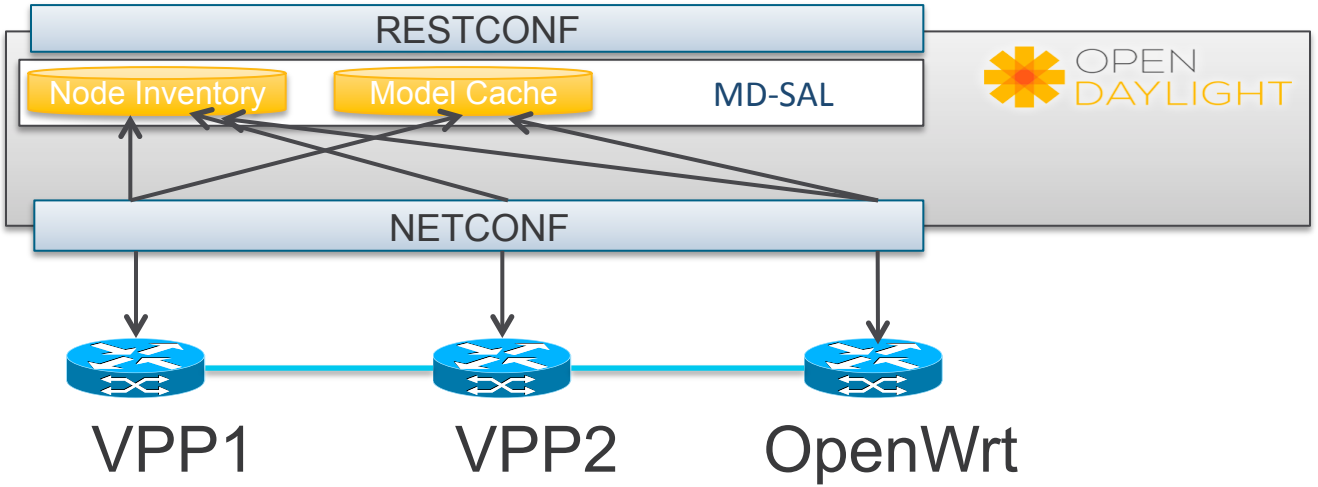
```
<node xmlns="urn:TBD:params:xml:ns:yang:network-topology">
  <node-id>vpp1</node-id>
  <host xmlns="urn:opendaylight:netconf-node-topology">{{vpp1_address}}</host>
  <port xmlns="urn:opendaylight:netconf-node-topology">2831</port>
  <username xmlns="urn:opendaylight:netconf-node-topology">admin</username>
  <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
</node>
```

High Level Manageability Architecture



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.



Installation

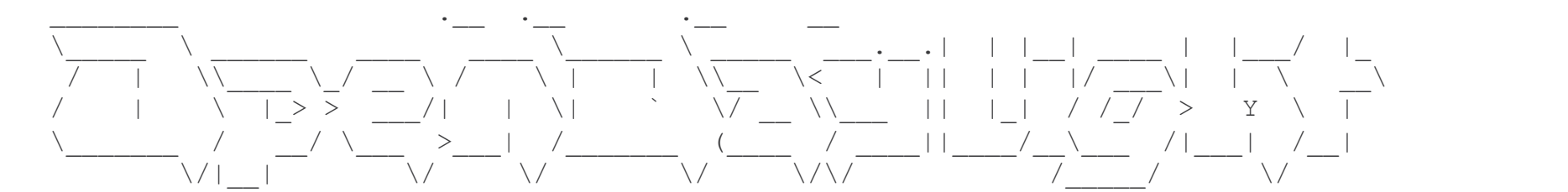
Distributions

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

Downloads

Release	Release date	Downloads	Documentation
Carbon SR2	October 16, 2017	• Pre-Built Tar • Pre-Built Zip • NeXT UI • Virtual Tenant Network (VTN) Coordinator	• Getting Started Guide • Developers Guide • User Guide • Installation Guide • Using OpenDaylight with OpenStack • Release Notes
Nitrogen SR1 (Current Release)	November 26, 2017	• Pre-Built Tar • Pre-Built Zip • Virtual Tenant Network (VTN) Coordinator • OpFlex	• Getting Started Guide • Developers Guide • User Guide • Installation Guide • Using OpenDaylight with OpenStack • Release Notes

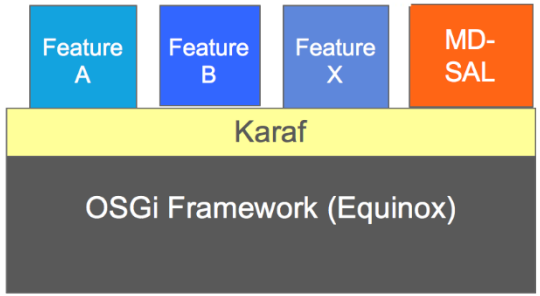
```
$ unzip karaf-0.7.1.zip
Archive:  karaf-0.7.1.zip
   creating: karaf-0.7.1/system/ …
$ cd karaf-0.7.1
$ ./bin/karaf
karaf: Enabling Java debug options: -Xdebug -Xnoagent -Djava.compiler=NONE
   -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005
Listening for transport dt_socket at address: 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

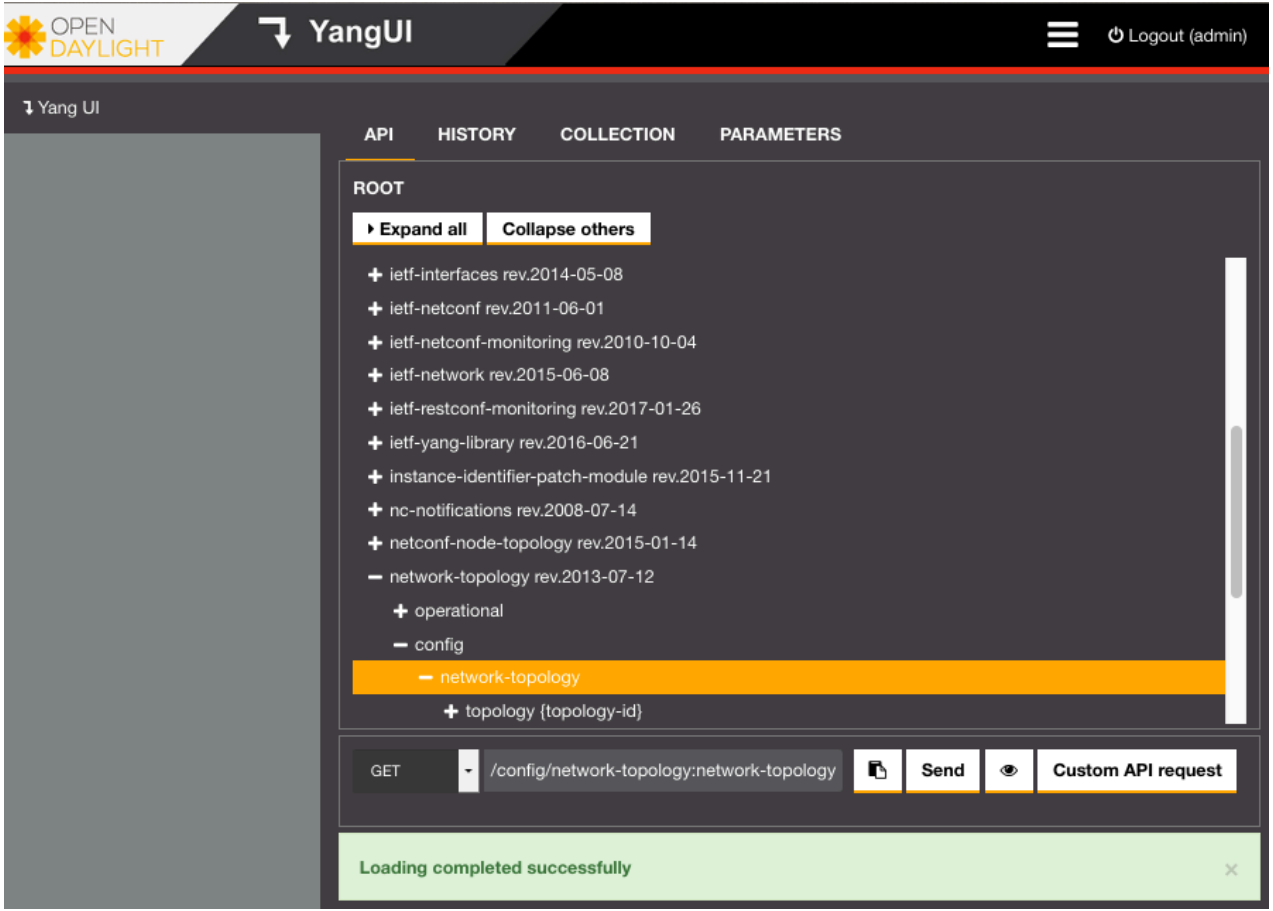
- OpenDaylight distro comes without any features enabled by default
- All features are available for you to install
  - *feature:list* — list all features available
  - *feature:list -i* — list all features installed
  - *feature:install <feature>* — install the <feature> feature
  - *feature:install <feature-1> <feature-2> … <feature-n>* — install list of features
  - *feature:uninstall <feature>* — uninstalls the <feature> feature



Install DLUX, NETCONF, and RESTCONF

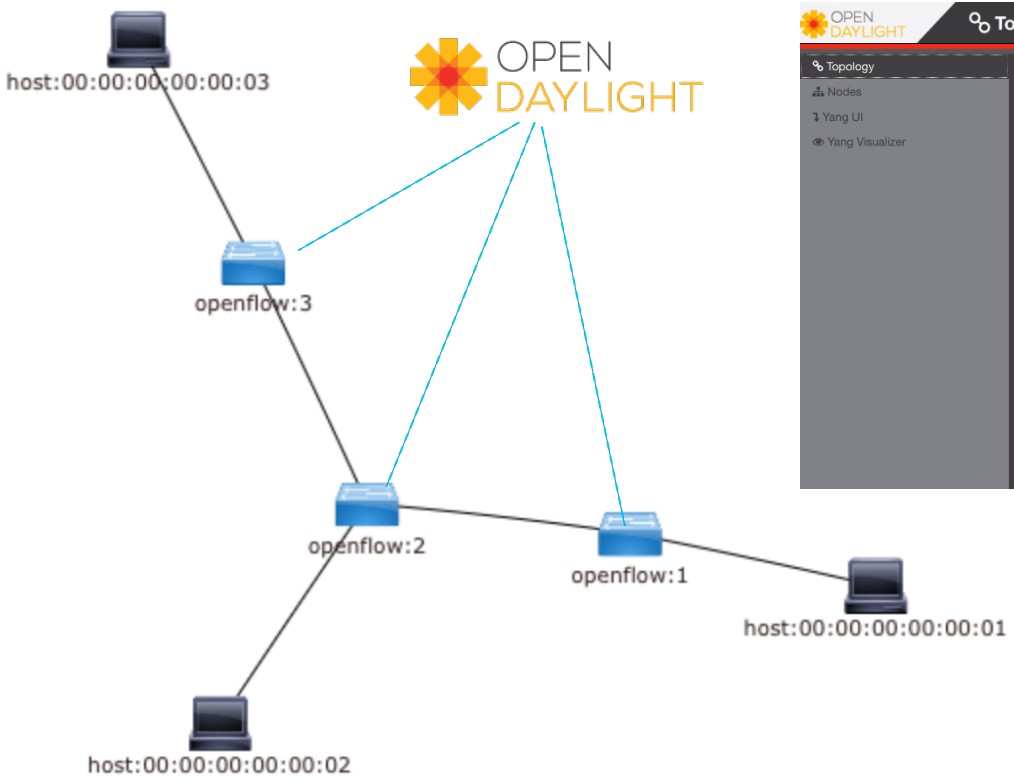
```
opendaylight_user@root> feature:install odl-dlux-core
opendaylight_user@root> feature:install odl-dluxapps-yangui
opendaylight_user@root> feature:install odl-restconf-all
opendaylight_user@root> feature:install odl-netconf-all
opendaylight_user@root> feature:install odl-netconf-topology
Opendaylight_user@root>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

Name                          | Version Required State
------------------------------------------------------------
odl-netconf-topology          | 1.3.1 | x        | Started
odl-restconf-all              | 1.6.1 | x        | Started
odl-netconf-connector-ssh     | 1.3.1 | x        | Started
odl-dluxapps-yangui           | 0.6.1 | x        | Started
odl-netconf-all               | 1.3.1 | x        | Started
odl-dlux-core                 | 0.6.1 | x        | Started
wrap                          | 0.0.0 | x        | Started
standard                      | 4.0.10| x        | Started
```

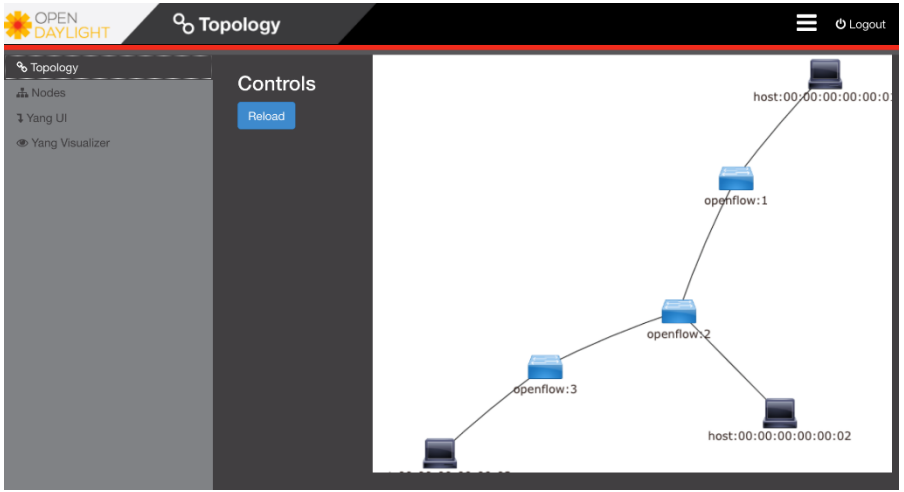
http://localhost:8181/index.html#/yangui/index



Example Use Cases

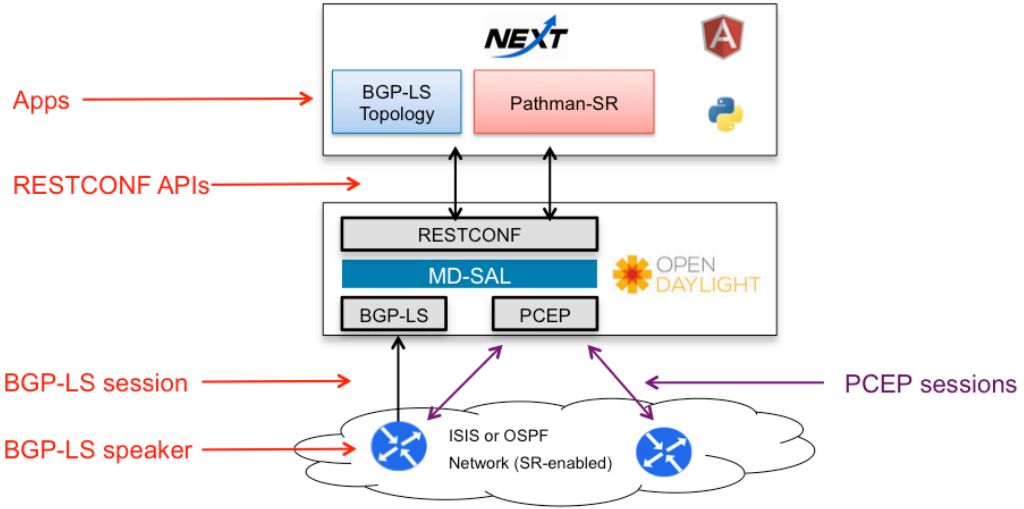
Mininet, OVSDB and OpenFlow





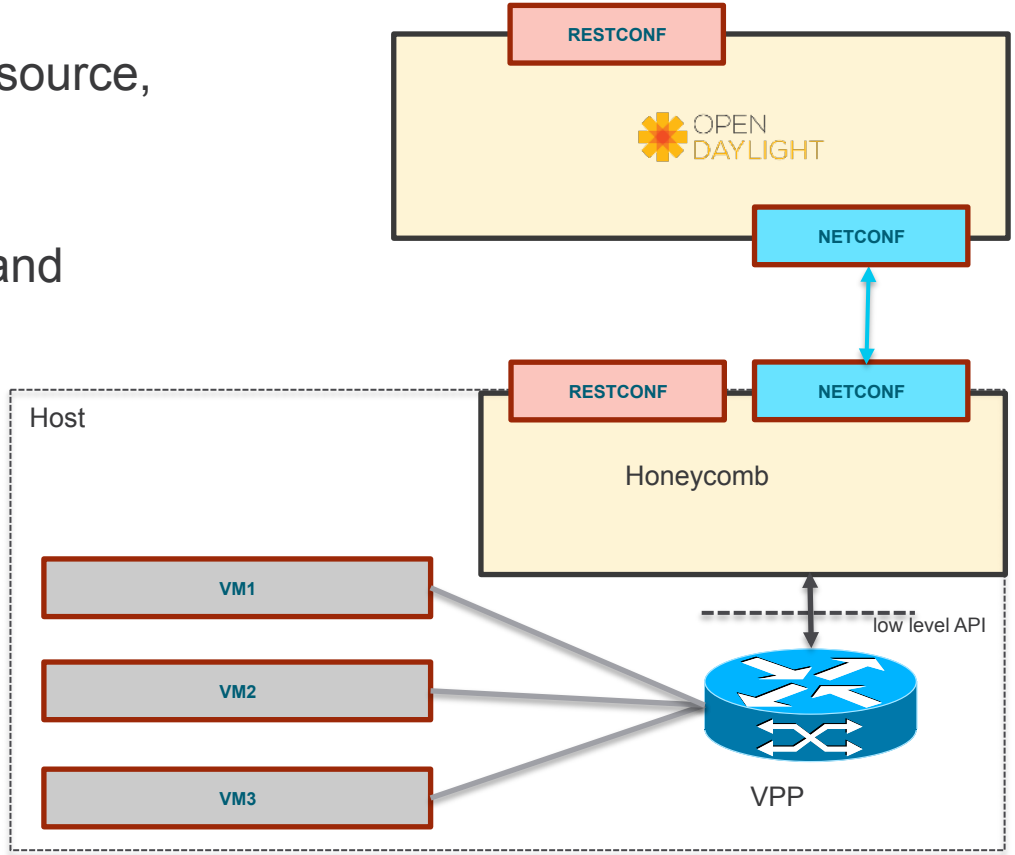
Cisco IOS XR using BGP-LS and PCE-P

- Cisco XRv topology in dCloud
  - dCloud is http://dcloud.cisco.com (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)
  - http://github.com/CiscoDevNet/opendaylight-setup
- Use Pathman-SR application to create Segment Routed LSPs
  - http://github.com/CiscoDevNet/pathman-sr

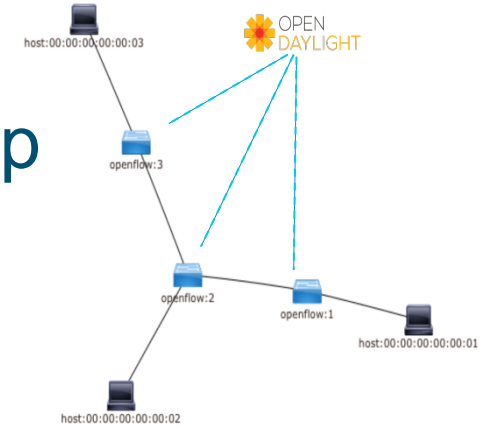


VPP/Honeycomb using NETCONF and RESTCONF

- VPP is a high-performance, open source, software forwarder
  - http://www.fd.io
- Honeycomb provides NETCONF and RESTCONF interfaces to VPP



OpenDaylight with Mininet – Step by Step



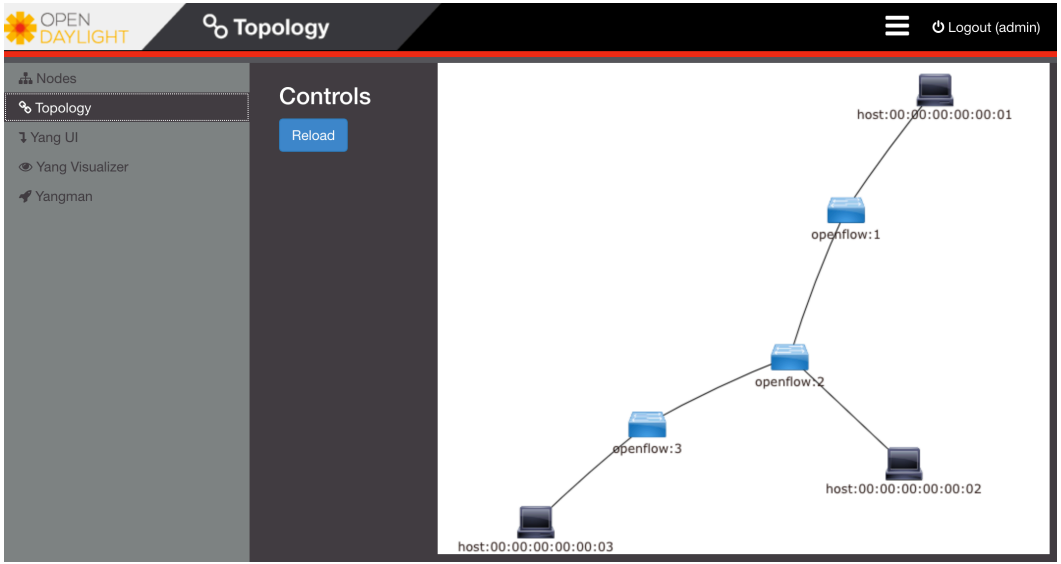
- Install, setup, and start Mininet VM using VirtualBox
  - Great instructions at http://www.brianlinkletter.com/set-up-mininet/
  - Login (user=mininet, password=mininet)
- Within OpenDaylight, enable required feature set
  - `opendaylight-user@root> feature:install odl-l2switch-switch odl-dlux-core 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
  - `http://<OpenDaylight-IP>:8181/index.html` (credentials: admin/admin)

Mininet Network Start

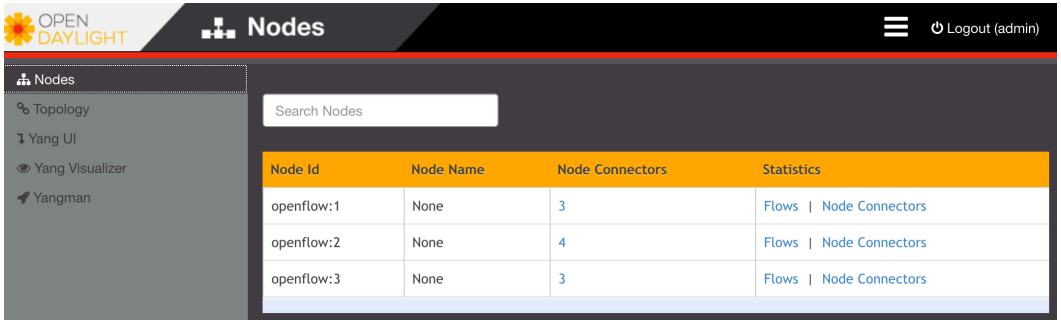
```
mininet@mininet-vm:~$ sudo mn --topo linear,3 --mac --controller=remote,ip=192.168.40.18,
port=6633 --switch ovs,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> 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)

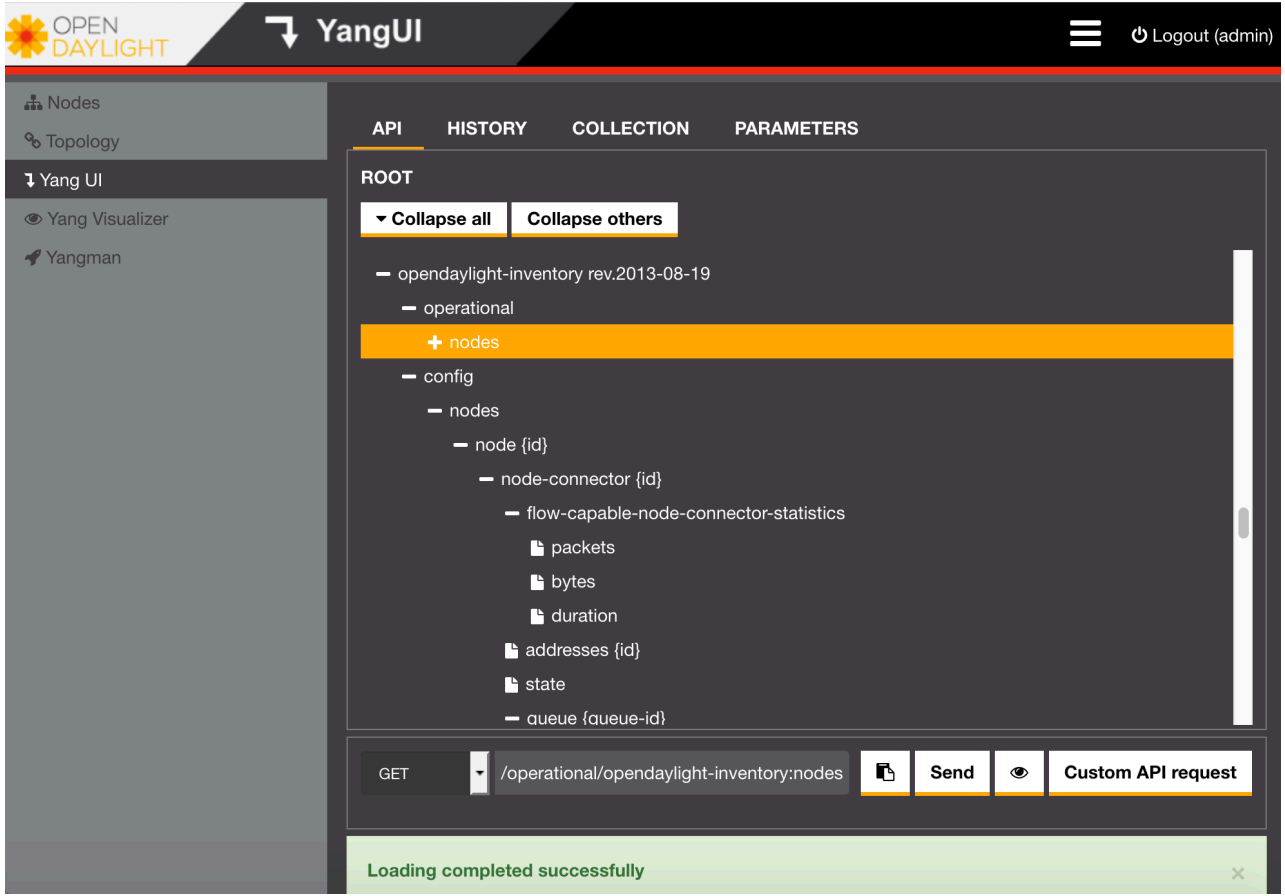


- Check out the network and switches by clicking on *Nodes*, *Node Connectors*



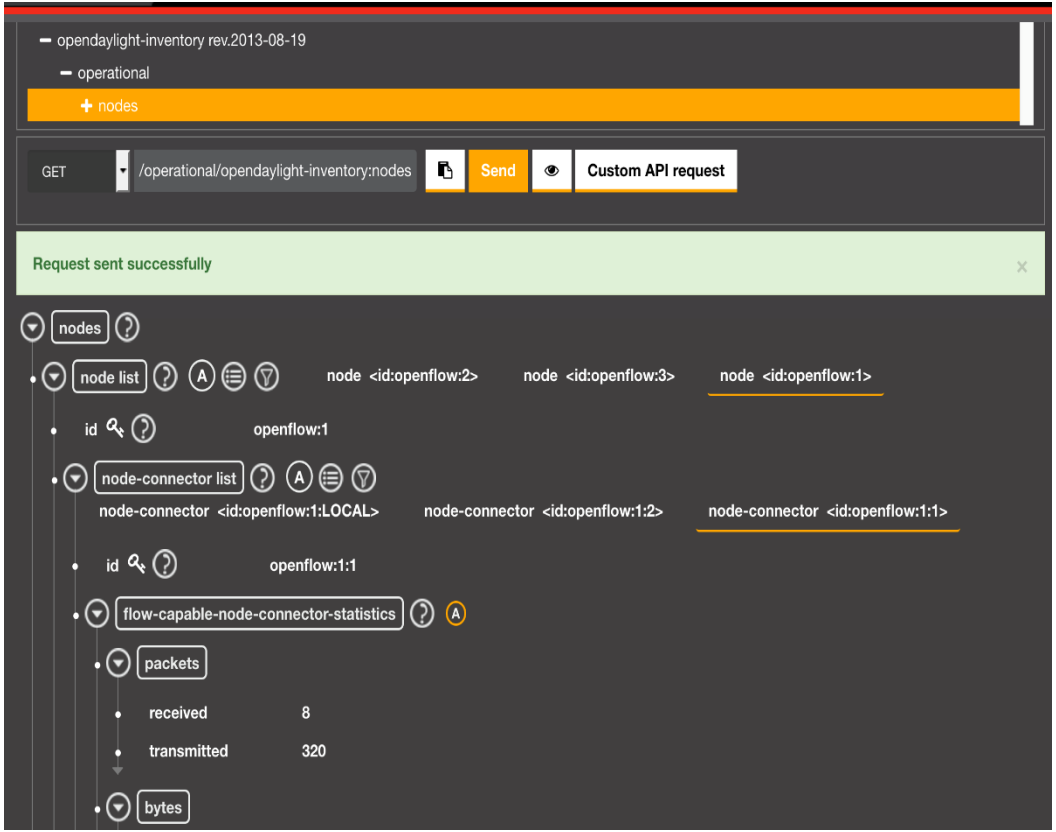
REST APIs

- Click on *Yang UI* and *Expand All* to see the REST APIs available



Inventory of Network Nodes

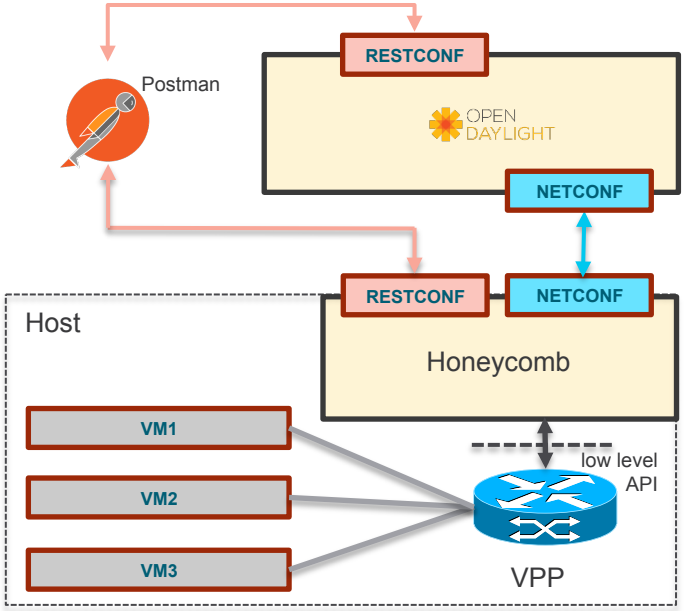
- GET opendaylight-inventory -> operational -> nodes



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

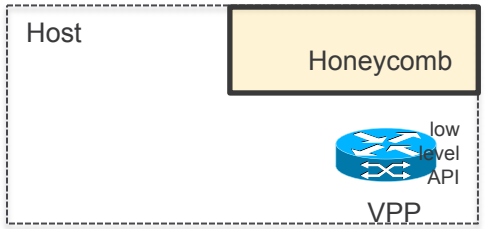
1. Create VM for Honeycomb and VPP

- 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

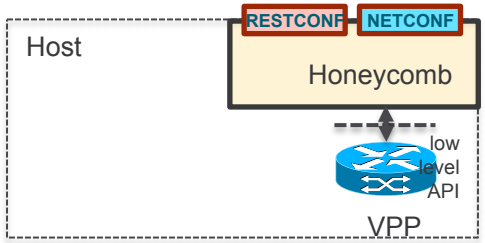
2. Install VPP and Honeycomb on VM

- 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

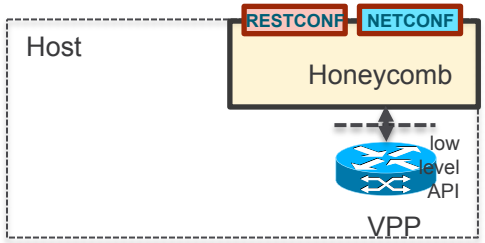
3. Start VPP and Honeycomb



- 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) https://wiki.fd.io/view/VPP/Command-line_Interface_(CLI)_Guide
  - $ ssh devnet@192.168.60.101
  - $ sudo vppctl

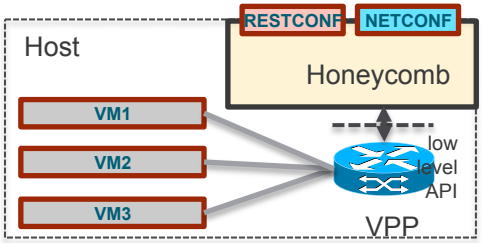
```
    _______    _        _ _____  ___
 __/ __/ _ \  (_)__    | | / / _ \/ _ \
 _/ _// // / / / _ \   | |/ / ___/ ___/
 /_/ /____(_)_/\___/   |___/_/  /_/
```

  - $vpp# show interface

Name	Idx	State
GigabitEthernet0/8/0	1	down
local0	0	down

VPP/Honeycomb using NETCONF and RESTCONF

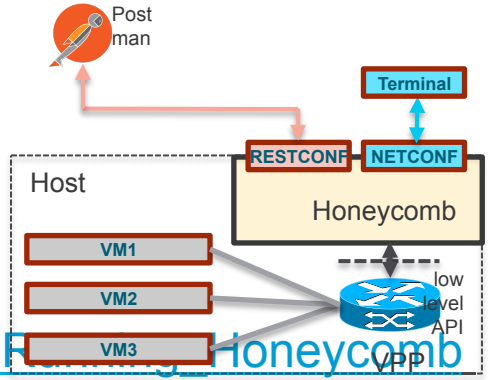
5. Add interface(s) to VPP



- 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

VPP/Honeycomb using NETCONF and RESTCONF

6. Connect to VPP Using Honeycomb and NETCONF

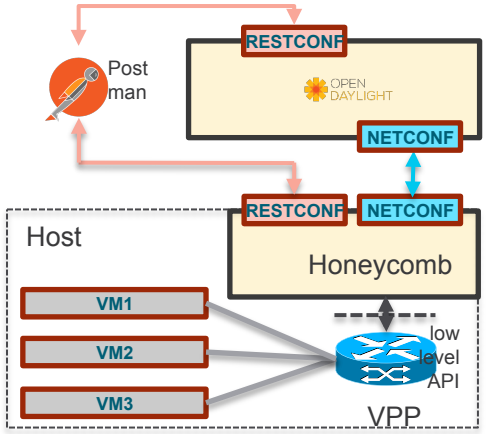


- Honeycomb listens on port 2831 for SSH/NETCONF
- Connect to VPP and issue for sample commands using: https://wiki.fd.io/view/Honeycomb/Releases/1609/
- You also need to add ssh-dss when connecting via ssh
  - $ ssh -oHostKeyAlgorithms=+ssh-dss admin@192.168.60.101 -p 2831 -s netconf
- By default, honeycomb 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",

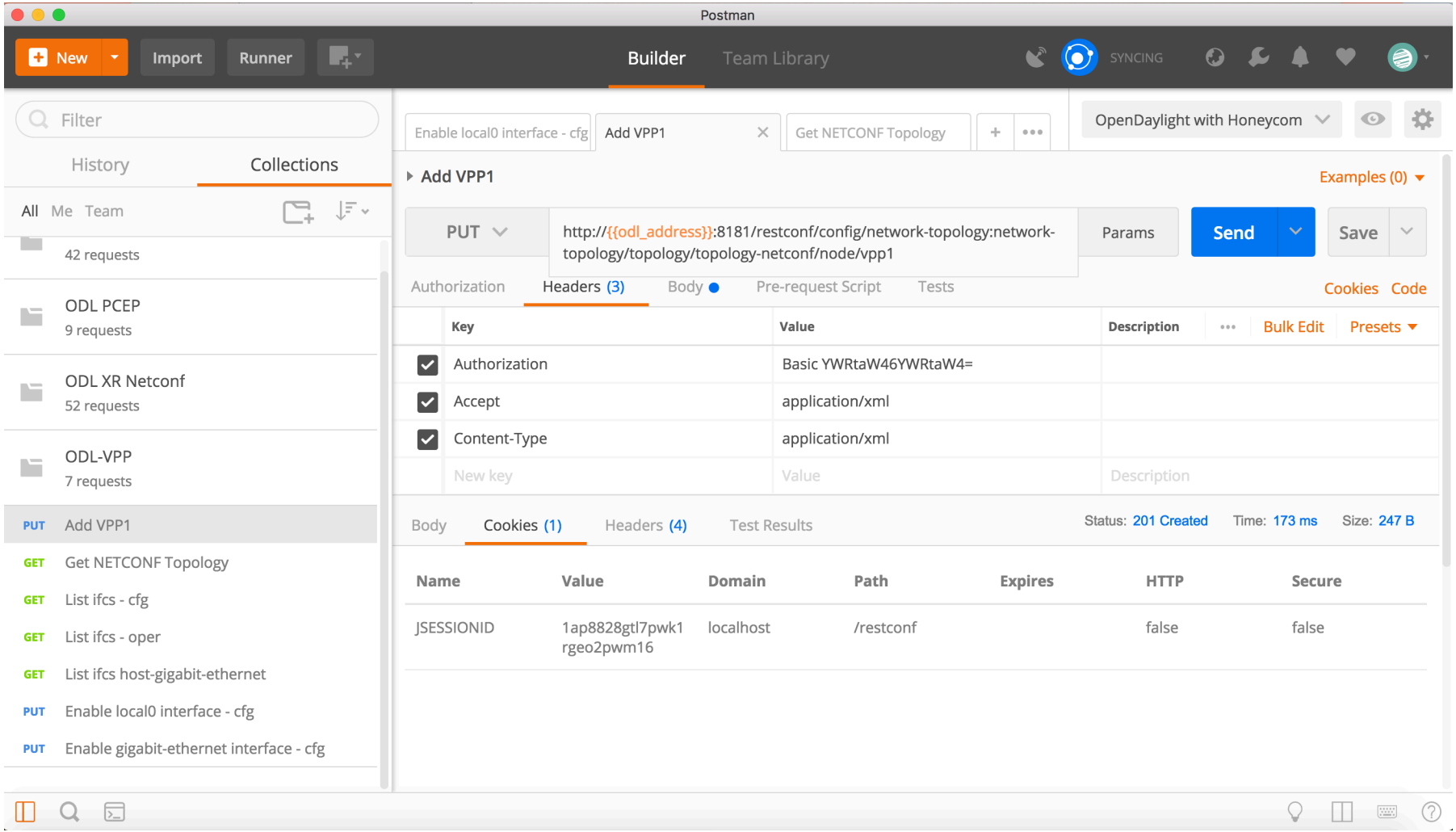
      "restconf-port": 8183,

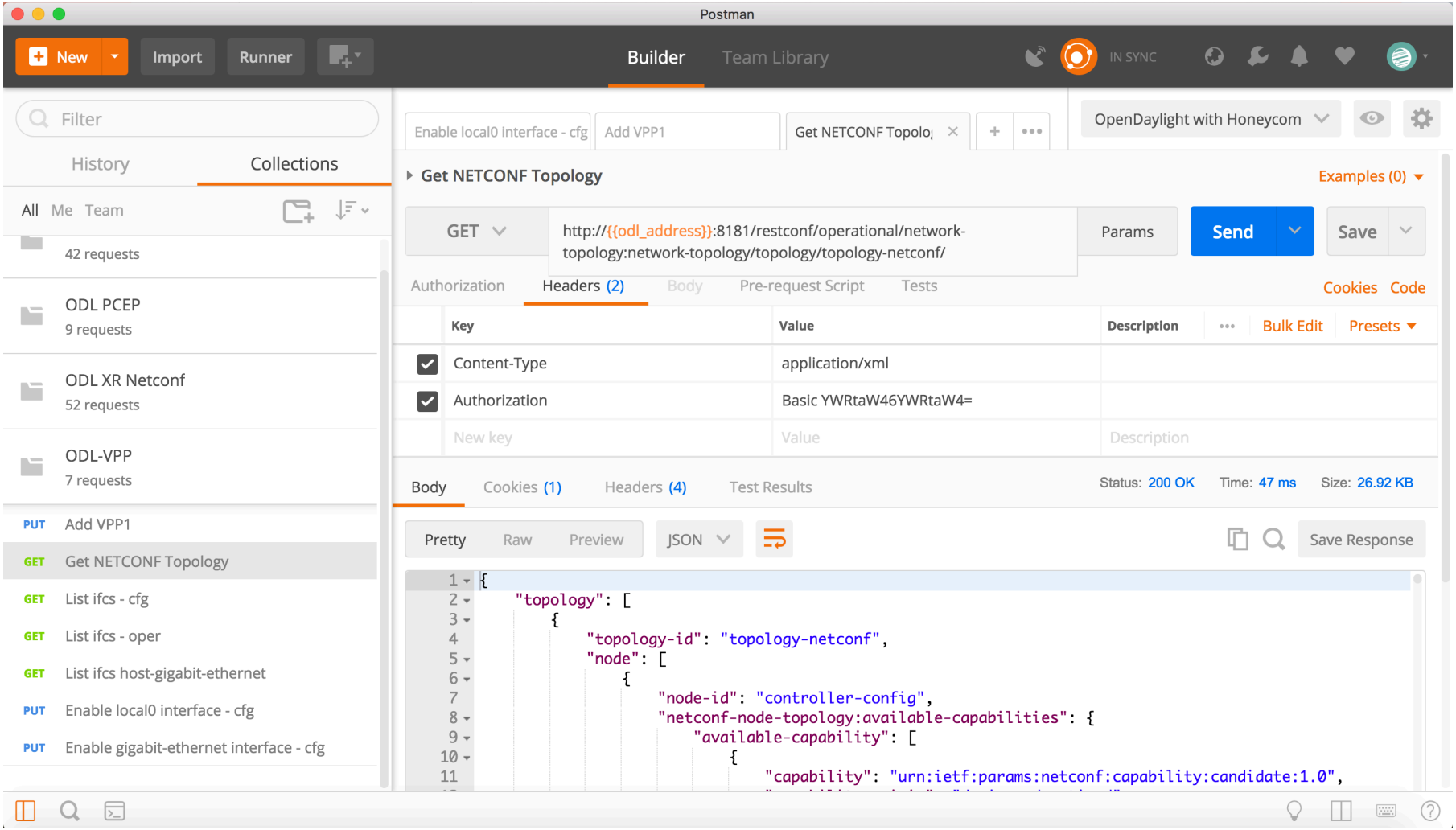
VPP/Honeycomb using NETCONF and RESTCONF

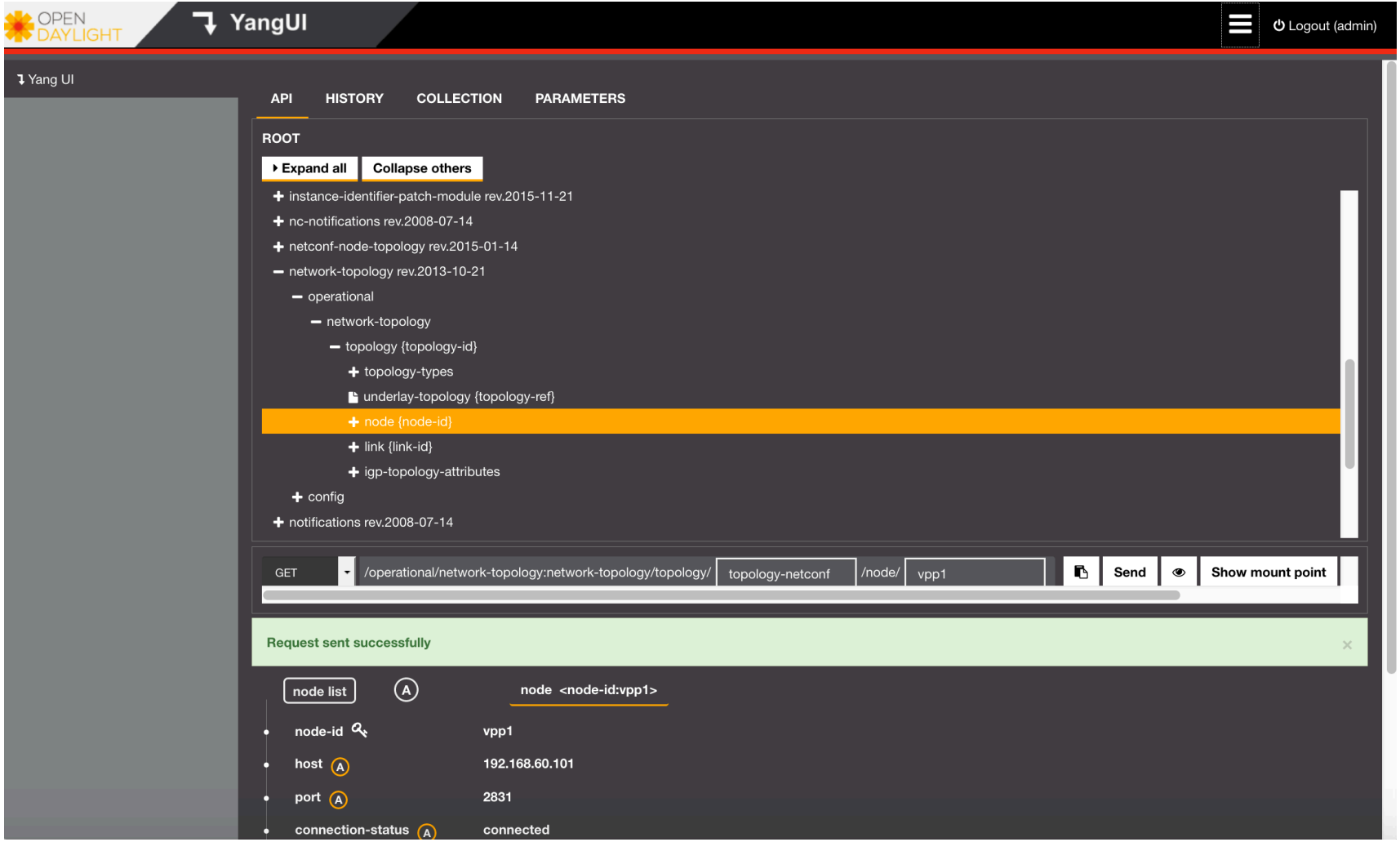


7. Connect to VPP Using OpenDaylight

- Import Postman environment
  - https://github.com/CiscoDevNet/opendaylight-sample-apps/blob/master/postman-collections/ODL-VPP-env.json
- Import Postman collection
  - https://github.com/CiscoDevNet/opendaylight-sample-apps/blob/master/postman-collections/ODL-VPP.json
- Add VPP to OpenDaylight topology with Postman
  - PUT http://{{odl_address}}:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/vpp1
- View configuration in OpenDaylight DLUX







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!