OpenDaylight
Current and Future Use Cases

Abhijit Kumbhare
OpenDaylight Technical Steering Committee (TSC) Chair
Principal Architect / System Manager, Ericsson
Agenda

• OpenDaylight Overview and Architecture

• OpenDaylight Use Cases (Partial List)
  I.  Network Abstraction
  II. ONAP
  III. Network Virtualization
  IV. AI/ML with OpenDaylight
  V.  ODL in OSS

• OpenDaylight: Getting Involved

• Acknowledgements

• Q & A
OpenDaylight Overview and Architecture
Past Two Days …

• Dinner Discussion with Phil Robb, VP of Operations, Networking & orchestration, Linux Foundation
  • Topic: our first OpenDaylight Meetings
  • November 2012

Nostalgic post by Dave Meyer, first ODL TSC chair on Facebook about first release Hydrogen in Jan 2014
Realization: We’re a bit old …

- As far as open source communities go – 6 years is like 60 dog years!!!

- But that’s great!!
  - We’ve got old timers
  - We’ve always been adding new developers
  - AND
OpenDaylight Project Goals

- **Code**: To create a robust, extensible, open source code base that covers the major common components required to build an SDN solution and create a solid foundation for Network Functions Virtualization (NFV)

- **Acceptance**: To get broad industry acceptance amongst vendors and users

- **Community**: To have a thriving and growing technical community contributing to the code base, using the code in commercial products, and adding value above and around.
OpenDaylight Now

- Mature, Open Governance
- 900 Contributors
- Over 100 deployments
- Multiple use cases
- Dozens of ODL-based solutions
- Mature code base – continued robust contributions even after 5+ years
- Focus on performance, scale and extensibility

https://opendaylight.biterg.io/
Service Abstraction Layer

• Initial SDN controllers
  • Controller application APIs strongly tied to OpenFlow
  • Hence applications developed limited to a single southbound protocol

• OpenDaylight Goal
  • Decouple the application API from the southbound protocol plugins - be that Openflow, NETCONF, OVSDB, PCEP, BGP, SNMP, or whatever.

• How to achieve the goal?
  • Use an abstraction layer – or what is called by OpenDaylight as Service Abstraction Layer or SAL
API Driven SAL (AD-SAL)

- Initial attempt at abstraction
  - API-Driven SAL, for communicating more directly with devices, using protocol(s) associated with the specific API.
- However abstraction difficult to realize in practice than it was in theory
  - AD-SAL became a collection of independent and discrete APIs, with one set of APIs for each and every southbound protocol
- AD-SAL was soon deprecated in OpenDaylight.
So how to achieve true abstraction?

- **Alternatives**
  - Build a better SAL
    - Take the existing APIs for the different plugins, and attempt to come up with an API abstraction that meets all of their needs
  - Use models
    - Implement a model layer within the SAL which has SDN applications dealing with software models of network devices, rather than directly with the devices themselves.
    - This was the approach taken by OpenDaylight – to develop a Model Driven SAL or the MD-SAL built around Yang models.
YANG

- Data modeling language that is also the preferred configuration language for NETCONF protocol

- Further reads:
  - YANG introductory tutorial
  - RFC 6020 - YANG - A data modeling language for NETCONF
  - RFC 7950 – The YANG 1.1 Data Modeling Language
What can YANG model?

- **Data**
- **RPCs:**
  - Perform procedure call with input/output, without worrying about actual provider for that procedure
- **Notifications:**
  - Publish one or more notifications to registered listeners
Applications built defining models
YANG used for defining models
Compilation results in the skeleton of application: model, RESTCONF API, etc.

Elements in red color above is the app skeleton
The model implementation (green) is where you will write code to do whatever it is that your application or the model within your application does
Yangtools – What does Yangtools do?

- Generates Java code from Yang
- Provides ‘Codecs’ to convert
  - Generated Java classes to Document Object Model (DOM)
  - DOM to various formats
    - XML
    - JSON
    - Etc
- ‘Codecs’ make possible automatic:
  - RESTCONF
  - Netconf
  - Other bindings
Yang to Java benefits

• Consistent Data Transfer Objects (DTOs) everywhere
  
  • **Automated Bindings:**
    • restconf
    • netconf
  
  • **Consistent:** reduce learning curve

• **Immutable:** to avoid thread contention

• **Improvable** – generation can be improved and all DTOs get those improvements immediately system wide
› Model-driven SAL is the kernel of the OpenDaylight controller

› It manages the contracts and state exchanges between every application. It does this adaptation by managing centralized state

› Takes in the YANG model at runtime and constructs the tree in the data store
OpenDaylight Architecture - Simplified View

Model-Driven Service Abstraction Layer (MD-SAL)

Controllers in a Cluster

App/Service

App/Service

Data

Notifications

RPCs

Plugin

Plugin

YANG Models
An Aspect of the architecture: ODL is a µ-services platform

Model-Driven SAL (MD-SAL)

OSS/BSS, External Apps

Network Devices

Netconf Server

RESTCONF

Application

Application

Protocol Plugin

Netconf Client

YANG-modeled interfaces

Messaging

“Kernel”

Microservices
OpenDaylight Architecture - Operational View

Third Party Applications (Orchestration, Control Plane, UI, etc.)

OpenDaylight APIs

Platform Services

Network Services And Applications

OpenDaylight Platform

Data Store (Config & Operational)

Messaging (Notifications / RPCs)

Interfaces & Protocol Plugins

Data Plane Elements (Virtual Switches, Physical Devices)

API

Protocol Plugin

Model
OpenDaylight Fluorine Release

Orchestration Applications

Control Plane Applications

Other Applications (e.g. Vendor UI)

OpenDaylight APIs (REST/RESTCONF/NETCONF)

Platform Services
- Authentication, Authorization and Accounting
- Data Export Import
- Infrastructure Utilities
- JSON-RPC Extension
- Time Series Data Repository

Network Services And Applications
- Container Orchestration Engine
- Genius Framework
- Honeycomb/Virtual Bridge Domain
- LISP Flow Mapping Service
- NEMO **
- Network Virtualization
- Neutron Service
- Service Function Chaining
- Transport PCE*
- Unified Secure Channel Manager **
- User Network Interface Manager

OpenDaylight Platform (Yangtools, MD-SAL)

Messaging (Notifications / RPCs)

Data Store (Config & Operational)

BGP BMP LISP NETCONF OpenFlow OVSDB PCEP SNMP SXP

Southbound Interfaces & Protocol Plugins

Data Plane Elements (Virtual Switches, Physical Device Interfaces)

* First release for the project
** Not included in Fluorine distribution - separate download
OpenDaylight Architecture: Key Takeaway

• OpenDaylight architecture is amenable to be applied to a variety of use cases as:
  • Not tied to a particular protocol
  • Modular, Extensible
  • Has built-in tools to simplify application development
OpenDaylight Use Cases (Partial List)
Note

• OpenDaylight architecture has been used in many use cases – not all covered here
Use Case I

Network Abstraction

Orchestration/OSS/Cloud plugin

Network Services API
(Path, Tunnel, L2/L3/L4 Service, Service Assurance, etc)

OpenDaylight

Control Interfaces
(OpenFlow, BGP, PCEP)

Management Interfaces
(Netconf, REST, OVSDB)

White Box Device

Traditional Network device

Provides Network Services API for Network Automation in a Multi Vendor Network
SDN-C & App-C based on OpenDaylight code
A set of projects working in tandem to provide network virtualization (overlay connectivity) inside and between data centers for Cloud SDN use case:
- VxLAN within the data center
- L3 VPN across data centers
- Integration with OpenStack Neutron and Kubernetes (in-progress)
- Uses Open vSwitch and hardware VTEPs (ToR) as the datapath
Network Virtualization: OpenDaylight Components

- **OpenDaylight NB APIs (REST)**
  - OF NSFs
    - Forwarding Rules Mgr
    - Inventory Mgr
  - OF NSFs
    - Neutron NB
    - Cardinal (SNMP)
  - AAA
  - Network NSFs
    - MP-BGP Interface
    - VPN Mgr
    - FIB Manager
    - NAT Service
    - IPv6 control service
  - ELAN Service
    - L2GW Handler
    - ACL service
    - QoS Service
    - DHCP Service
  - ODL GENIUS
    - Interface Manager
    - Internal Transport Manager
    - Liveness Manager
    - Lock Manager
    - MD-SAL datastore
    - YANG tools
    - Notification broker
    - Clustering
    - DAEXIM

- **Model-Driven service abstraction layer (MD-SAL) (plug-in mgr., capability abstractions, ...)**

- **ODL Platform**
  - BGP Protocol Engine (Quagga)

- **Legend**
  - ODL GENIUS
  - ODL Netvirt
  - ODL Infrastructure
  - Misc Services
  - External module
A common controller platform

One Application / Service

Containerized Network Functions  Interconnect  Virtual Network Functions  Interconnect  Hardware Appliances

- Containerized Network Functions
- Virtual Network Functions
- Hardware Appliances

- kubernetes
- openstack
- Opendaylight

- DCGW
- Fabric
- Fabric NMS
- BGPVPN
- EVPN
- OVSDB

- Uniform service capabilities
- Simplified interworking
- Reduced training and validation
- Simplified troubleshooting
- Common dashboard
OpenDaylight multi-instance controller

Containerized applications on per tenant hosted K8s

Openstack VM’s

VNF VNF VNF

Tenant K8s (VM’s) Tenant K8s (VM’s) Tenant K8s (VM’s)

Kuryr CNI Kuryr CNI Kuryr CNI

Neutron OpenDaylight driver

Openstack (containerized)

Infra Kubernetes (bare metal)

OpenDaylight CNI
OpenDaylight Container Orchestration Engine

**Current Status**

- Hybrid scenario:
  - Openstack and Kubernetes side by side
  - Integration with ODL via Openstack Kuryr
  - Supports Multinode environment
  - Supports container in a VM scenario

- Baremetal scenario
  - Kubernetes only
  - Tight integration with ODL NetVirt
  - Supports Pod 2 Pod networking L2/L3

**Future Scenarios**

- Support for non-OF southbound
  - NetConf
- Testing with L3VPN for multi-tenant scenarios
- Scale testing & improvement
Use Case IV (future)
AI/ML with OpenDaylight

Smart SDN Controller

- Network status awareness
  - Rely on time series data collected from the network
- Traffic Control Policy Change decision making
  - Based on the advanced analytics and machine learning.
- Dynamic change of Control policies
  - Automatically change the traffic control policies based on the analytics results.

Advanced Analytics & Machine Learning

Automated Traffic Control

Time Series Data Collection
Why we need Machine Learning in SDN

• Software Defined Networks needs to be intelligent.
  • To be aware of the runtime status of the network.
  • To make the right decisions that adjust the policies for traffic classification and traffic shaping.
  • To dynamically change the policies according to the analytics results.
    • AI / MI can be used to establish normalized profiles and dynamically update the profiles based on a set of predetermined or dynamically learned rules.
Use Cases of a smart and intelligent SDN controller

➢ Traffic Control and Routing Optimization
  • Congestion Control
  • Traffic Pattern Prediction
  • Routing Optimization

➢ Resource optimization
  • Networking resource allocation optimization
  • Cloud resource management optimization

➢ Security and Anomaly Detection
  • DDoS attack detection and mitigation

➢ Troubleshooting and Self-healing
AI/ML Example Use Case – Traffic congestion prediction with automated control

1. Collect stats from the network and store into TSDR
2. Data analysis through data analytics engines integration
3. Traffic flow redirection from A->F to A->B->F and A->D->E->F

SDN controlled network

- Predicted congestion path in the next 24 hours
- Healthy path in the next 24 hours
Prediction using Weka leveraging data collected in TSDR
Enable AI/ML on both historical and real-time data paths.

Many use cases would require both offline and online ML on the time series data.

External events could be additional input for accurate machine learning results.

Feed back the results to SDN control path for automatic traffic steering and policy placement.

Well-defined interface among the components towards future standardization of advanced analytics in SDN.
ODL AI/ML framework PoC Architecture

- PoC of both historical offline machine learning and real-time online machine learning
  - Collect the time series data
  - Persist into scalable data storage
  - Publish to high performance data bus
- Integrate with external machine learning libraries
  - Spark MLlib
  - DeepLearning4J
- Collect OpenFlow Stats and apply machine learning algorithms
  - k-means clustering
WAN Transport Orchestrator (WAN-O)

- Based on ACTN (Abstraction of Control of Traffic Engineered Network) IETF Standard for realizing hierarchical SDN architecture
  - Yang Based (NetConf/RESTCONF) Models
Coordination of resources across multiple independent networks and multiple technology layers to provide end-to-end services

Layered operational model:

- **Customer**: issuing a service request from catalog
- **Service Provider**: dealing w/ Customer and providing the service (may or may not own the network(s) as such)
- **Network Provider**: infrastructure providers owning the physical network(s) and building the infrastructure

SDN Hierarchical architecture based on ACTN

- CNC – Customer Network Controller
- MDSC – Multi Domain Service Coordinator
- PNC – Provisioning Network Controller

- CMI – CNC-MDSC Interface
- MPI – MDSC-PNC Interface
- SBI – South Bound Interface
WAN-O as MDSC, interfaces

MDSC NBI:
- CMI: CNC to MDSC interface
- YANG based (Netconf/Restconf)
- End to end Virtual Network concept
- Unified end to end topology

MDSC SBI:
- MPI: MDSC to PNC interface
- YANG based (Netconf/Restconf)
- Per domain TE-Tunnels
- White or Black Domain topology
Transport Network architecture

Service Orchestration (Operator 1)

- IETF ACTN MPI
  - White topology

- IETF ACTN MPI
  - Black topology

SDNc Operator 1

SDNc Operator 2

SDNc Operator 3

Inter domain link

White topology domain

Black topology domain
END to END service orchestration
Connectivity services

1. Service Orchestration
2. WAN Transport SDN (Underlay)
END to END service orchestration
VNF services

1. Service Orchestration
2. WAN Transport SDN (Underlay)
3. Network Virtualization (Overlay)
OpenDaylight: Getting Involved
Avenues for getting involved

• OpenDaylight Wiki: https://wiki.opendaylight.org
• Mailing Lists:
  • Central / Cross Project: https://wiki.opendaylight.org/view/Mailing_Lists
  • Complete List including individual projects: https://lists.opendaylight.org/mailman/listinfo
• Chat with developers via IRC: https://wiki.opendaylight.org/view/IRC
• Meetings:
  • Technical Steering Committee: https://wiki.opendaylight.org/view/TSC:Meeting
  • Technical Work Stream: https://wiki.opendaylight.org/view/Tech_Work_Stream:Main
  • Complete List including individual projects: https://wiki.opendaylight.org/view/Meetings
Areas to getting involved in

- OpenDaylight Documentation Project
- Project of your interest
  - [https://wiki.opendaylight.org/view/Project_list](https://wiki.opendaylight.org/view/Project_list)
  - Code Reviews
  - Bug Fixing
- MD-SAL & Clustering (Distributed Systems)
  - Experts
  - Enthusiasts: Improve your skills in these hot & in-demand area
- Scale & Performance
- Testing
- Architecture Improvements
  - Example: Scalable and Robust Data Replication using etcd.
Acknowledgements
- Contributors to slides
  - Antonio De Gregorio
  - Colin Dixon
  - Daniele Ceccarelli
  - Dayavanti Kamath
  - Francois Lemarchand
  - Frederick Kautz
  - Jan Medved
  - Luis Gomez
  - Prem Sankar Gopanan
  - Scott Melton
  - Srini Seetharaman
  - YuLing Chen

- Reference
  - https://github.com/BRCDcomm/BVC/wiki/MD-SAL
Q & A