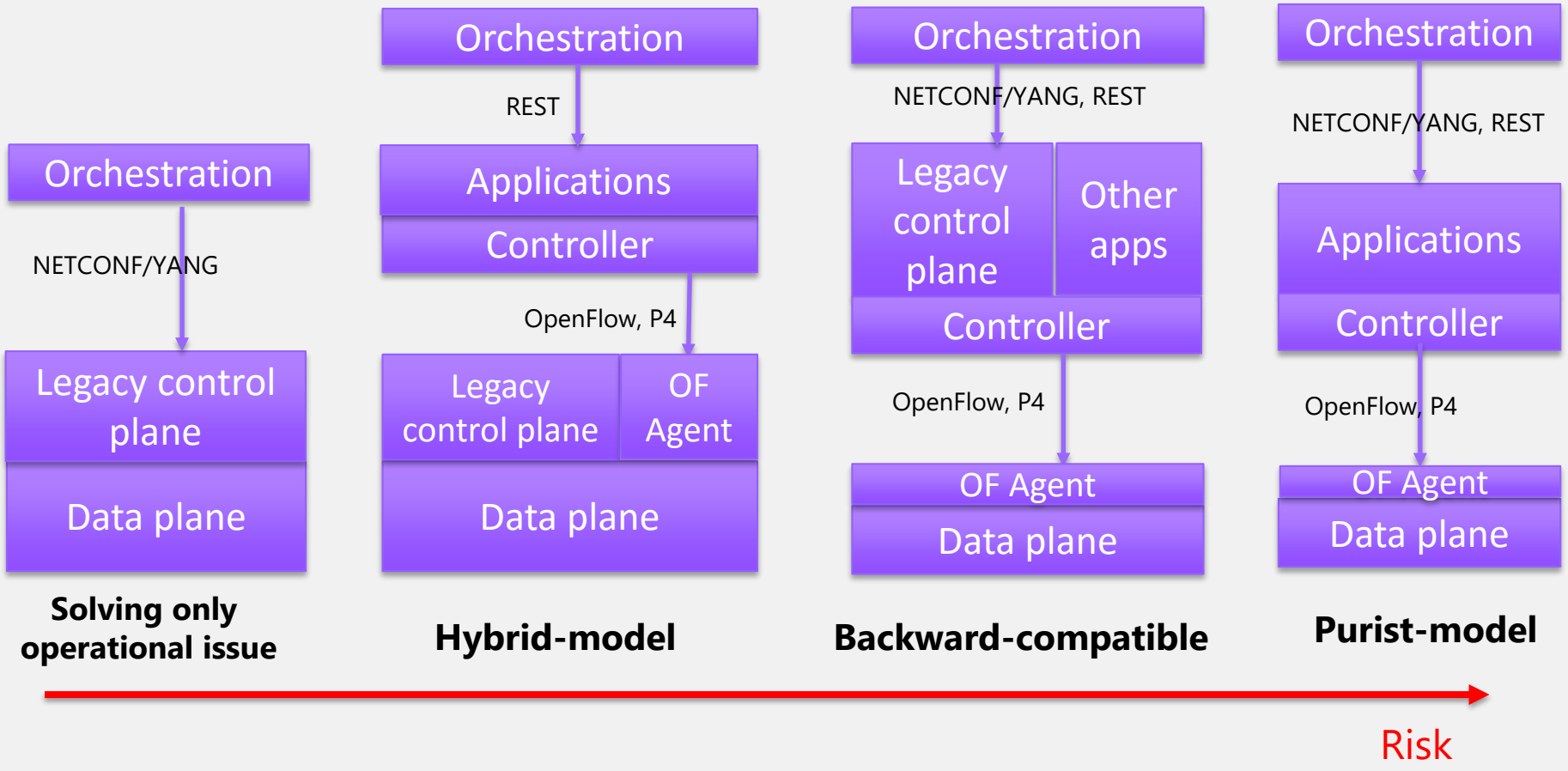
The Software in SDN

Programming in Opendaylight

Presenter's Note

This presentation borrows heavily from a presentation on App Development by Srini Seetharaman and are available at http://sdnhub.org

OpenDaylight embraces the full spectrum of choices



Lithium: List of Projects (Total: 40)

Project keyword	Description
aaa	AAA Project
alto*	Application-Layer Traffic Optimization
bgpcep	BGP/PCEP protocol library
capwap*	Control and Provisioning of Wireless Access Point
controller	OpenDaylight Controller
defense4all	Radware Defense4All
didm*	Device Identification and Drive Management
discovery	Discovery Service
dlux	OpenDaylight UI
docs	Documentation Project
groupbasedpolicy	Group Based Policy Plugin
integration	Integration Framework
iotdm*	IoT Data-centric Middleware
l2switch	Separate Layer 2 switching
lacp*	Link Aggregation Control Protocol
lispflowmapping	LISP Mapping Service
nic*	Network Intent Composition
openflowjava	OpenFlow Protocol Library
openflowplugin	OpenFlow Plugin
opflex*	OpFlex protocol
ovsdb	OVSDB protocol and OpenStack integration
packetcable	PacketCable PCMM/COPS
pss*	Persistence Store Service
plugin2oc	Southbound plugin to the OpenContrail platform
reservation*	Dynamic Resource Reservation project
sdninterfaceapp	SDNi Cross-controller interface
sfc	Service Function Chaining
snbi	Secure Network Bootstrap Infrastructure
snmp4sdn	SNMP4SDN Plugin
snmp*	SNMP Southbound Plugin
sxp*	Source-Group Tag eXchange Protocol
tcpmd5	TCP-MD5 Support library
toolkit	Quickstart Toolkit
tpf*	Topology Processing Framework
tsdr*	Time Series Data Repository
ttp	TTP Project
usc*	Unified Secure Channel
vtn	VTN (Virtual Tenant Network)
yangtools	YANG Tools

* → New in Lithium release

OPEN DAYLIGHT

4th Release "Beryllium" Production-Ready Open SDN Platform

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

AAA AuthN Filter

OpenDaylight APIs REST/RESTCONF/NETCONF/AMQP

Base Network

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

Enhanced Network Services

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

Network Abstractions (Policy/Intent)

- 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)

OU SUPERCOMPUTING CENTER FOR EDUCATION & RESEARCH
OSCER
UNIVERSITY OF OKLAHOMA

What does it mean to program in ODL?

1. Get familiar with the Model-View-Control approach for app development in a modular fashion
   - YANG (Yet Another Next Generation) Model for data, RPC and notifications
   - RESTconf View generated automatically
   - Implementation in Java to handle data changes, notifications and RPC call backs

2. Get familiar with platform essentials (maven, config subsystem, dependencies) and useful tools

3. Learn about existing projects and reuse modules
   - No need to change code of other projects. Just link them as binary libraries

YANG modeling

YANG

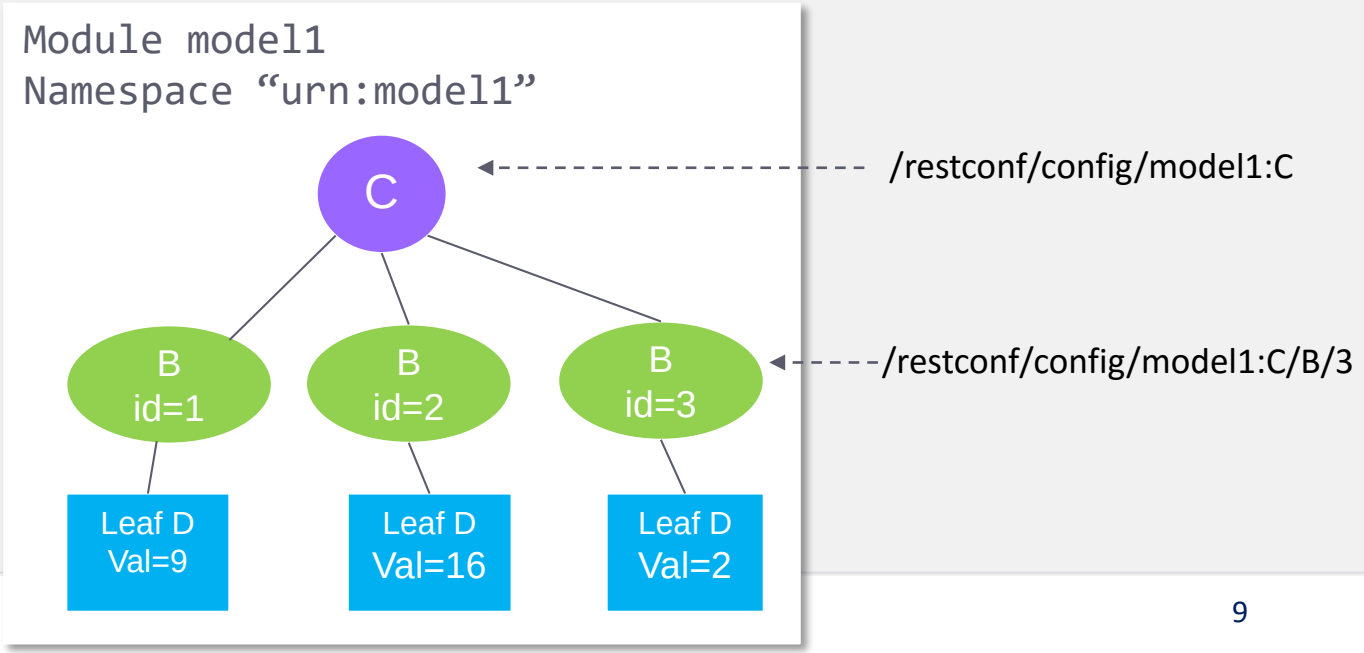
- Data modeling language that is also the preferred configuration language for NETCONF protocol

- Further reads:
  - YANG introductory tutorial
  - RFC 6020 - YANG - A data modeling language for NETCONF

```
module model1 {

    namespace "urn:model1";
    prefix model1;
    yang-version 1;

    revision 2015-04-06 {
        description "Initial revision"
    }

    grouping A {
        list B {
            key id;
            leaf id {
                type uint32;
            }
            leaf D {
                type uint32;
            }
        }
    }

    container C {
        uses A;
    }
}
```

MD-SAL Data Access

- 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
  - For the YANG model in previous slide, here is the view of the root and its children



MD-SAL Data Access (contd.)

- When the model is compiled with maven, you will see classes generated in Model1Data. java, B.java, and C.java

- InstanceIdentifier is used as a pointer to a child. Following points to the first child node in the figure

```
InstanceIdentifier iid = InstanceIdentifier.builder(C.class)
     .child(B.class, new BKey((long)1))
     .build();
```

- ReadOnlyTransaction, and WriteTransaction are used to access the data store.

```
B updatedB = new BBuilder().setD((long)19).build();
WriteTransaction modification = dataBroker.newWriteOnlyTransaction();
modification.merge(LogicalDataStoreType.CONFIGURATION, iid, updatedB, true);
modification.submit();
```

- Transaction can also be batched

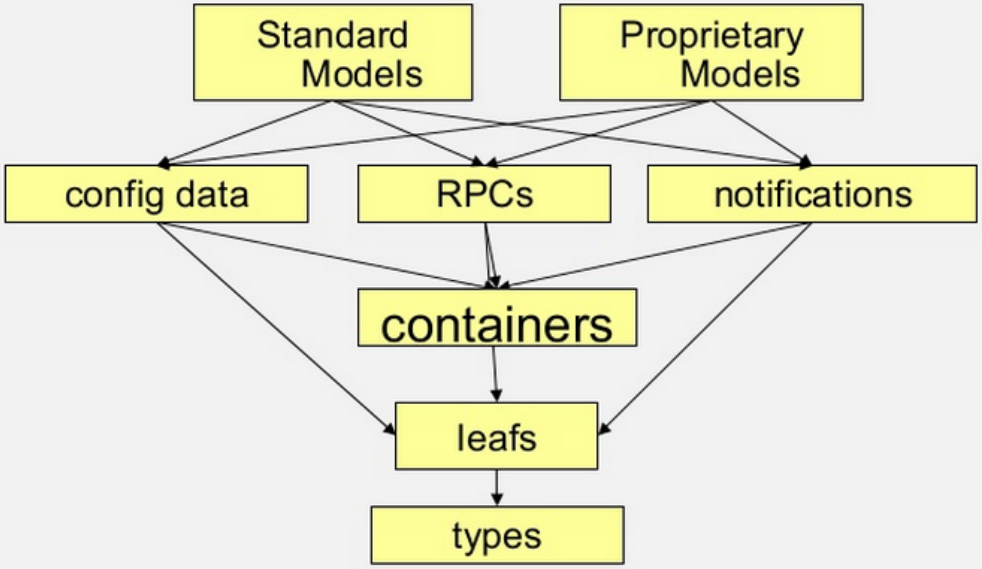
Two types of data store

- For the same model, there are two types of data store maintained by MD-SAL
  - Config store: App developers typically use this to store user input and associated derived state for apps
  - Operational store: Typically used to keep transient ephemeral state

- Choice of store has implications on RESTconf and persistence
  - Config store is always kept persistent. There is control on how many replicas to keep
  - Operational store cannot be changed over RESTconf

YANG not restricted to Just Data Store

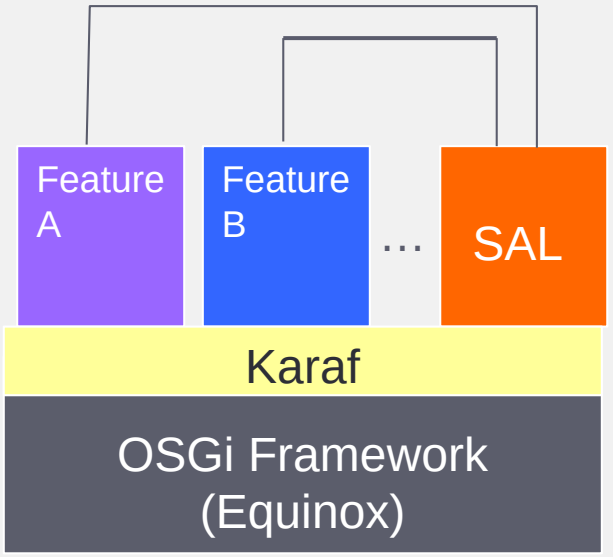
- Notifications:
  - Publish one or more notifications to registered listeners
- RPC:
  - Perform procedure call with input/output,
    without worrying about actual provider for that procedure



Poking into the basic platform

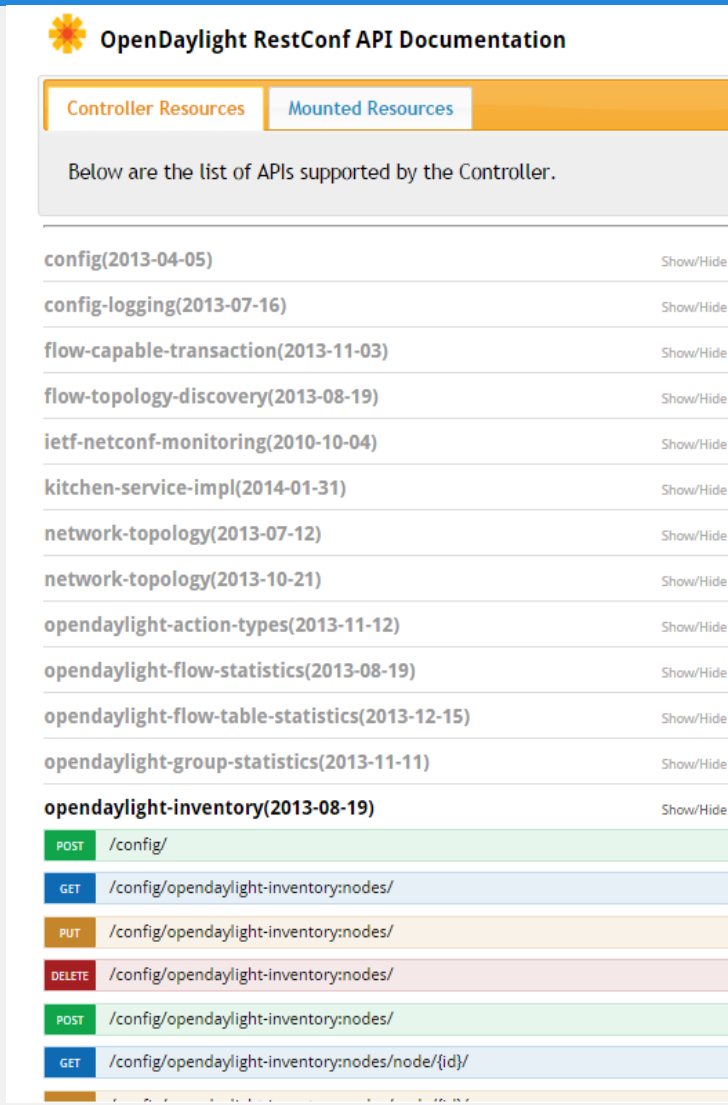
Java, Interface, Maven, OSGi, Karaf

- Java chosen as an enterprise-grade, cross-platform compatible language
- Java Interfaces are used for event listening, specifications and forming patterns
- Maven – build system for Java
- OSGi:
  - Allows dynamically loading bundles
  - Allows registering dependencies and services exported
  - For exchanging information across bundles
- Karaf: Light-weight Runtime for loading modules/bundles
  - OSGi based. Primary distribution mechanism for Helium



REST APIs

- OpenDaylight has significant support for REST APIs
- Restconf allows for checking config and operational state
  - `feature:install odl-restconf`
  - http://localhost:8181/restconf/....
- List of exposed Northbound APIs are autogenerated using swagger.
  - `feature:install odl-mdsal-apidocs`
  - http://localhost:8181/apidoc/explorer



ODL's opendaylight-inventory.yang (Lithium)

```
module opendaylight-inventory {
    namespace
"urn:opendaylight:inventory";
    prefix inv;

    typedef node-id {
        type inet:uri;
    }

    typedef node-connector-id {
        type inet:uri;
    }

    typedef node-ref {
        type instance-identifier;
    }

    typedef node-connector-ref {
        type instance-identifier;
    }

    grouping node {
        leaf id {
            type node-id;
        }

        list "node-connector" {
            key "id";
            uses node-connector;
        }
    }

    grouping node-connector {
        leaf id {
            type node-connector-id;
        }
    }

    container nodes {
        list node {
            key "id";
            uses node;
            . . .
        }
    }

    notification node-updated {
        status deprecated;

        leaf node-ref {
            type node-ref;
        }
        uses node;
    }

    notification node-connector-updated {
        status deprecated;

        leaf node-connector-ref {
            type node-connector-ref;
        }
        uses node-connector;
    }

    notification node-removed {
        status deprecated;

        leaf node-ref {
            type node-ref;
        }
    }

    notification node-connector-removed {
        status deprecated;

        leaf node-connector-ref {
            type node-connector-ref;
        }
    }
}
```

Augmentation made by OpenFlow plugin for storing:

1. All switch description
2. All OpenFlow features
3. All tables and its flows
4. All meters, groups

ODL's Inventory Config Data Store

http://localhost:8181/restconf/config/opendaylight-inventory:nodes

ODL's network-topology.yang (Beryllium)

http://localhost:8181/restconf/operational/network-topology:network-topology

```
module network-topology  {
    namespace
"urn:TBD:params:xml:ns:yang:network-topology";

typedef topology-ref {
        type leafref {
            path "/network-
topology/topology/inet:uri";
        }
    }

    typedef node-ref {
        type leafref {
            path "/network-
topology/topology/node/inet:uri";
        }
    }

    typedef link-ref {
        type leafref {
            path "/network-
topology/topology/link/inet:uri";
        }
    }

    grouping tp-attributes {
        leaf tp-id {
            type inet:uri;
        }
        leaf-list tp-ref {
            type tp-ref;
        }
    }
```

```
grouping node-attributes {
        leaf node-id {
            type inet:uri;
        }
        list supporting-node {
            key "node-ref";
            leaf node-ref {
                type node-ref;
            }
        }
    }
grouping link-attributes {
        leaf link-id {
            type inet:uri;
        }
        container source {
            leaf source-node {
                mandatory true;
                type node-ref;
            }
            leaf source-tp {
                type tp-ref;
            }
        }
        container destination {
            leaf dest-node {
                mandatory true;
                type node-ref;
            }
            leaf dest-tp {
                type tp-ref;
            }
        }
        list supporting-link {
            key "link-ref";
            leaf link-ref {
                type link-ref;
            }
        }
    }
```

```
container network-topology {
        list topology {
            key "topology-id";
            leaf topology-id {
                type inet:uri;
            }
            container topology-types {
            }
            list underlay-topology {
                key "topology-ref";
                leaf topology-ref {
                    type topology-ref;
                }
            }

            list node {
                key "node-id";
                uses node-attributes;
                list termination-point {
                    key "tp-id";
                    uses tp-attributes;
                }
            }

            list link {
                key "link-id";
                uses link-attributes;
            }
        }
    }
}
```

Extension to models through Augments

- Unless specified otherwise, YANG models can be augmented in a new namespace to add extra data within an existing data model.
- For example, odl-l2-switch augments above inventory to store hosts learned

```
import opendaylight-inventory { prefix inv; revision-date 2013-08-19; }

augment "/inv:nodes/inv:node/inv:node-connector" {
    ext:augment-identifier "address-capable-node-connector";
     uses address-node-connector;
}
```

```
curl http://localhost:8080/restconf/operational/opendaylight-
inventory:nodes/node/openflow:1/node-connector/openflow:1:1
{ "node-connector":
    [ { "id": "openflow:1:1",
        "address-tracker:addresses": [ {
            "address-tracker:last-seen": 1404918398057,
            "address-tracker:ip": "10.0.0.1",
            "address-tracker:first-seen": 1404918392926,
            "address-tracker:mac": "f2:0c:dd:80:9f:f7"
        }]
    }]
}
```

Code Design

- yangtools generates data model classes
  - Builders for generating immutable objects also generated
  - identity mapped to interfaces
  - All else are mapped to classes
  - Identity is the only one that allows inheritance

- Developers have to write the gateway adapters
  - Converting from one data model to another
  - Translating between a non-ODL source and ODL app

https://wiki.opendaylight.org/view/YANG_Tools:YANG_to_Java_Mapping

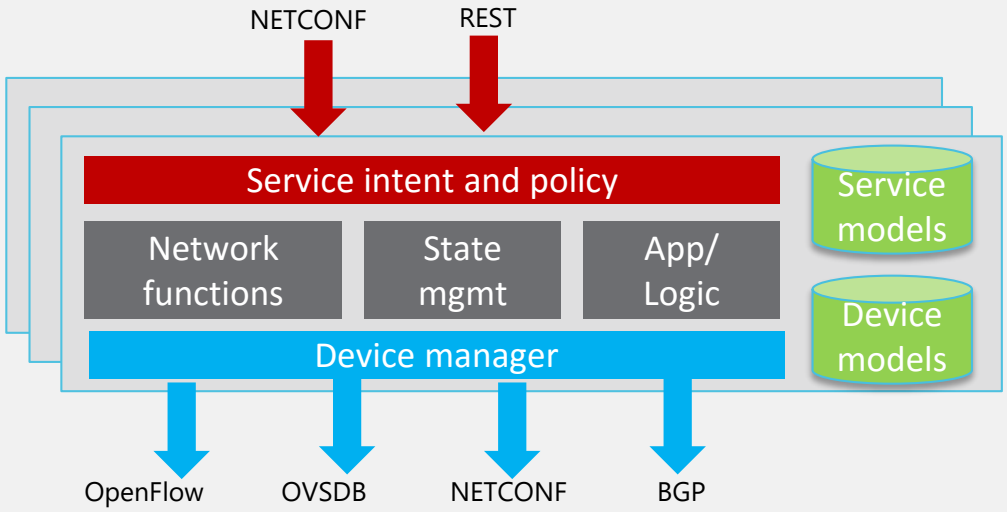
Data Store Management

- Persistence
  - Everything that gets into a shard is stored in-memory and on the disk.
  - Restarting the controller will reconstitute the state of the shards from the persisted data.
- Clustering
  - Multiple controller instances can interwork and share state (for backup reasons rather than load-sharing)
  - One leader (writer) and multiple followers (read-only) where state consistency is achieved using RAFT algorithm
- Sharding
  - The big in-memory MD-SAL tree is broken up into a bunch of smaller sub-trees such that 1 model is 1 shard (e.g., inventory, topology and default).

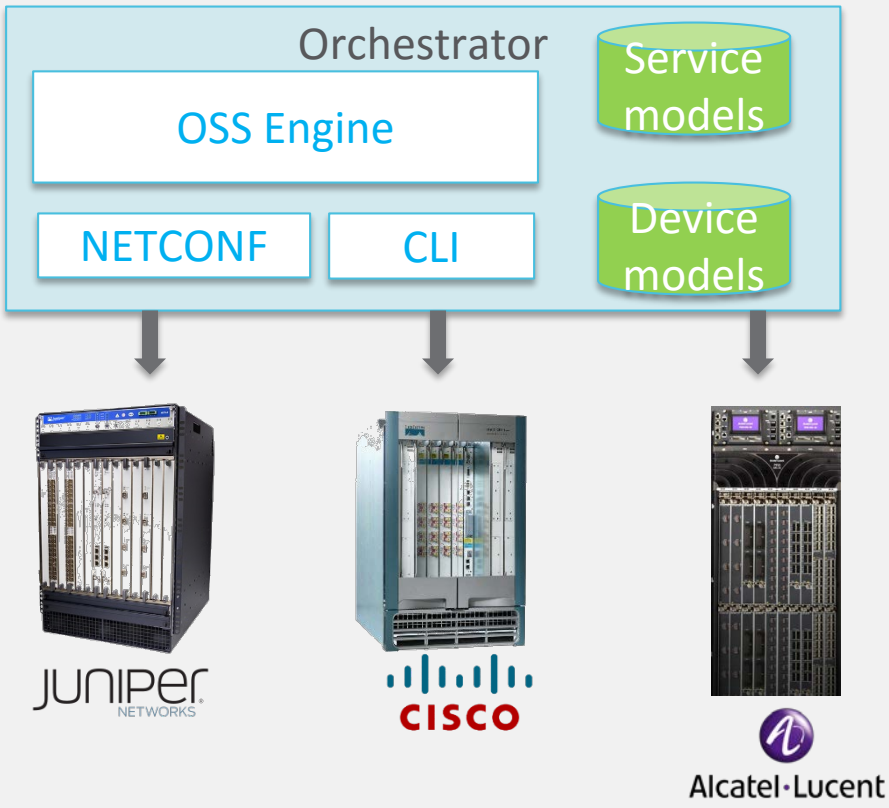
Southbound Plugins: OpenFlow and NETCONF

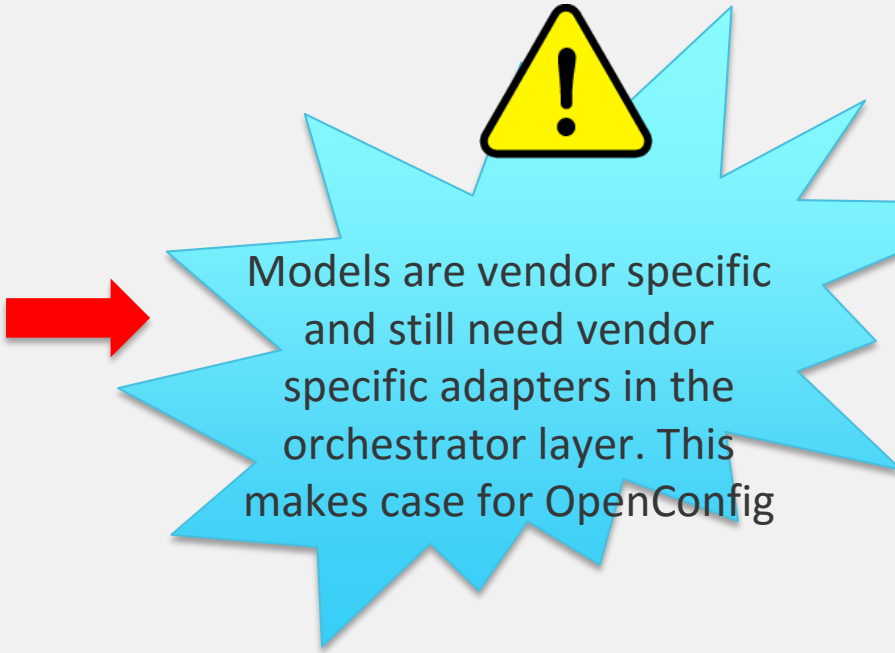
OpenDaylight External Interfaces

- Southbound:
  - Built-in NETCONF client and BGP speaker to interact with legacy control planes and orchestrate them
  - Plugin for OpenFlow 1.0 and 1.3 support
  - Plugin for OVSDB that works with OVS schema (VTEP schema in future)
- Northbound:
  - REST and NETCONF for receiving intent data



NETCONF based Multi-Vendor OSS





Questions?