The Software in SDN

Programming in Opendaylight
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

- **Orchestration**
  - REST
  - NETCONF/YANG

- **Applications**
  - OpenFlow, P4

- **Controller**
  - Legacy control plane
  - OF Agent

- **Data plane**

**Legacy control plane**

**Solving only operational issue**

**Hybrid-model**

**Backward-compatible**

**Purist-model**

**Risk**
### Lithium: List of Projects (Total: 40)

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

* → New in Lithium release
**Service Abstraction Layer/Core**

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

Enhanced Network Services
- AAA
- Centinel – Streaming Data Hdr
- Controller Shield
- Dev Discovery, ID & Dev Mgmt
- DOCSIS Abstraction
- Link Aggregation OAM 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 (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)

Messaging (Notifications / RPCs)

OpenDaylight APIs REST/RESTCONF/NETCONF/AMQP

Data Plane Elements (Virtual Switches, Physical Device Interfaces)

Southbound Interfaces & Protocol Plugins
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
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”

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

```
Module model1
Namespace “urn:model1”

C

B
id=1

B
id=2

B
id=3

Leaf D
Val=9

Leaf D
Val=16

Leaf D
Val=2

/restconf/config/model1:C

/restconf/config/model1:C/B/3
```
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:

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

ReadOnlyTransaction, and WriteTransaction are used to access the data store:

```java
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)

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

```yang
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;
}
}
```

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

```
notification node-updated {
  status deprecated;
  leaf node-ref {
    type node-ref;
  }
}
```

```
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;
  }
  uses node;
}
```

```
notification node-connector-removed {
  status deprecated;
  leaf node-connector-ref {
    type node-connector-ref;
  }
  uses node-connector;
}
```
ODL’s Inventory Config Data Store

http://localhost:8181/restconf/config/opendaylight-inventory:nodes
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 OVSDDB that works with OVS schema (VTEP schema in future)

► Northbound:
  ▪ REST and NETCONF for receiving intent data
NETCONF based Multi-Vendor OSS

Models are vendor specific and still need vendor specific adapters in the orchestrator layer. This makes case for OpenConfig.
Questions?