OPEN DAYLIGHT
FORUM INDIA 2015

# Getting Started with OpenDaylight

Natarajan Dhiraviam, Kalaiselvi K, Dell R&D

OpenDaylight can do for networking what Linux has done for the computing industry.

- David Meyer of Brocade

Background to get started with ODL

OPEN DAYLIGHT
FORUM INDIA 2015

# Overview of ODL

# Agenda

- Brief Insight into ODL Architecture
- Introduction to SAL / MD –SAL
- Introduction to SAL plugins / flows
- Data modelling and YANG
- Karaf
- Maven and Build
- Test environment

OPEN DAYLIGHT FORUM INDIA 2015

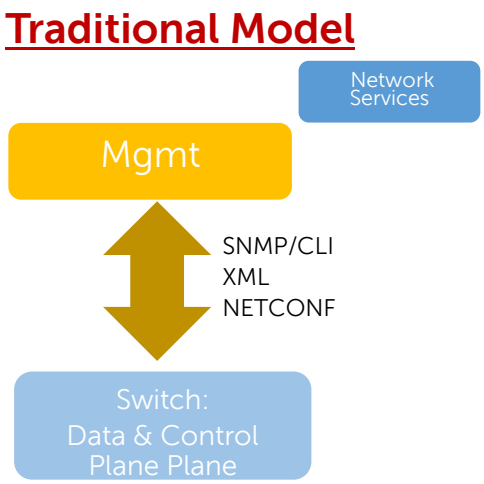
# Background

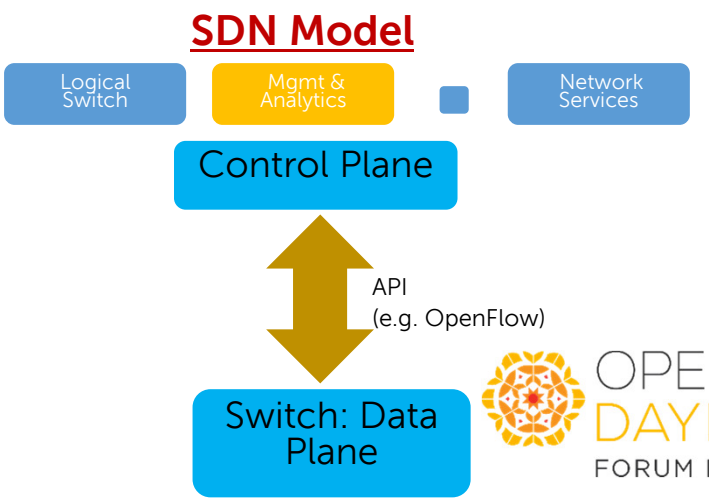
# What is SDN?

- According to Open Networking Foundation [ Purist definition ]

SDN is a **new approach** to networking in which **network control is decoupled** from the **data forwarding** function and is **directly programmable**.

The result is an extremely **dynamic**, **manageable**, **cost-effective**, and **adaptable** architecture that gives administrators **unprecedented programmability, automation, and control**.

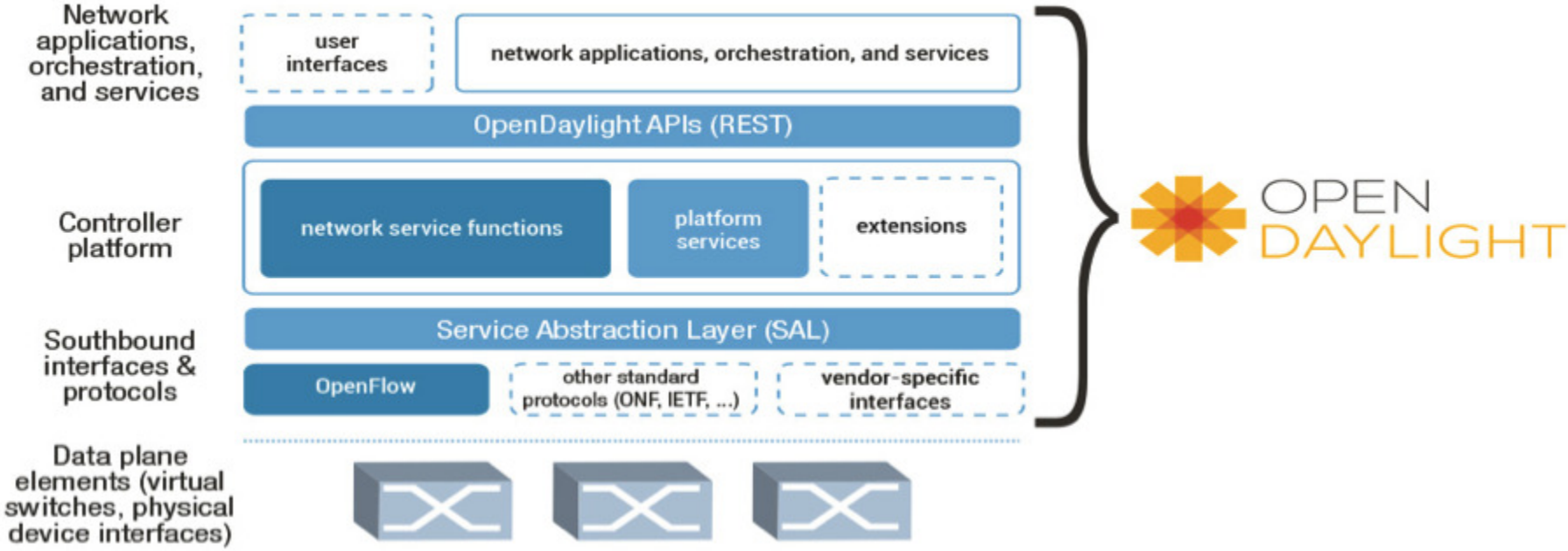


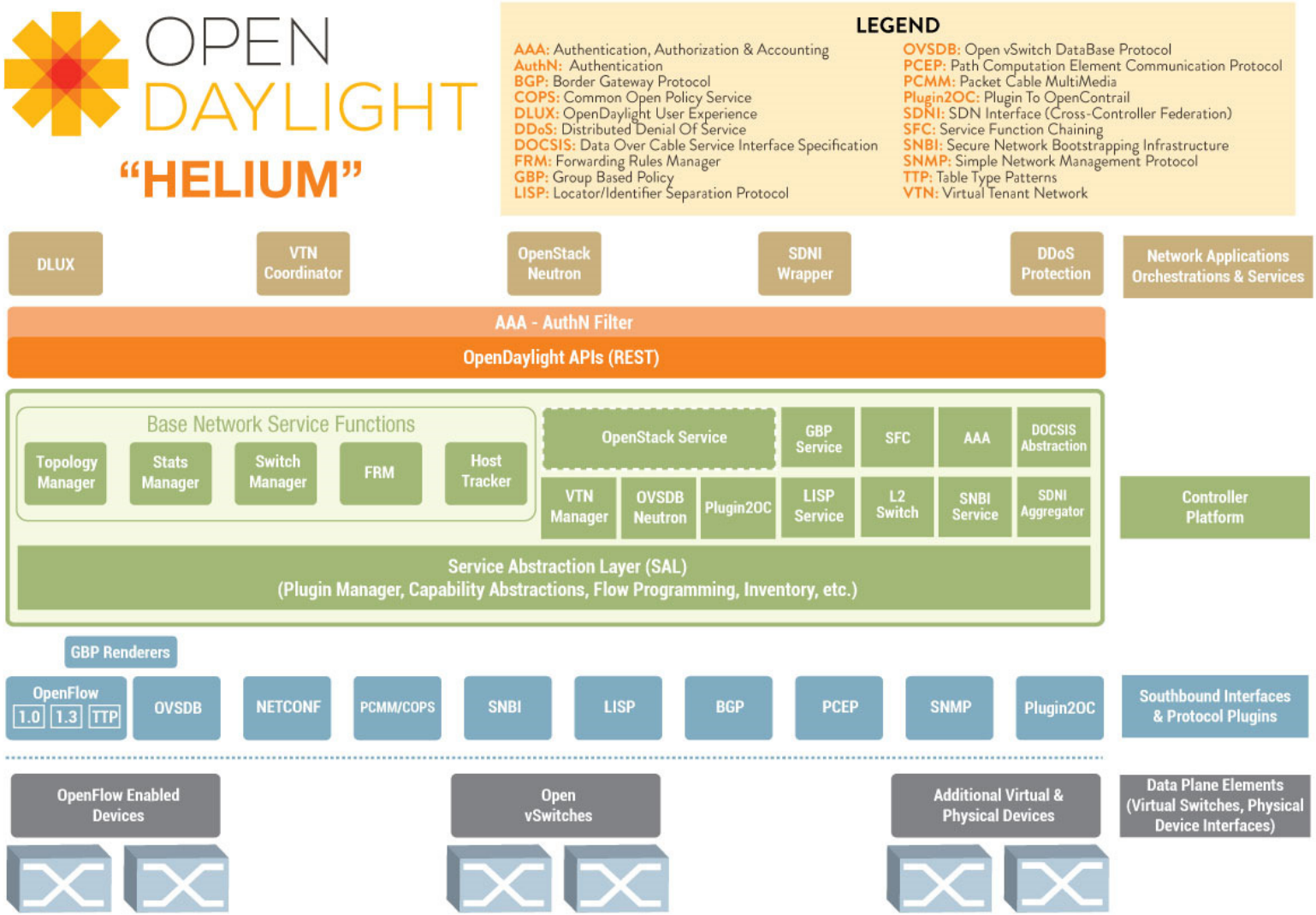

OPEN DAYLIGHT
FORUM INDIA 2015

# ODL

# ODL Structure



# ODL Components

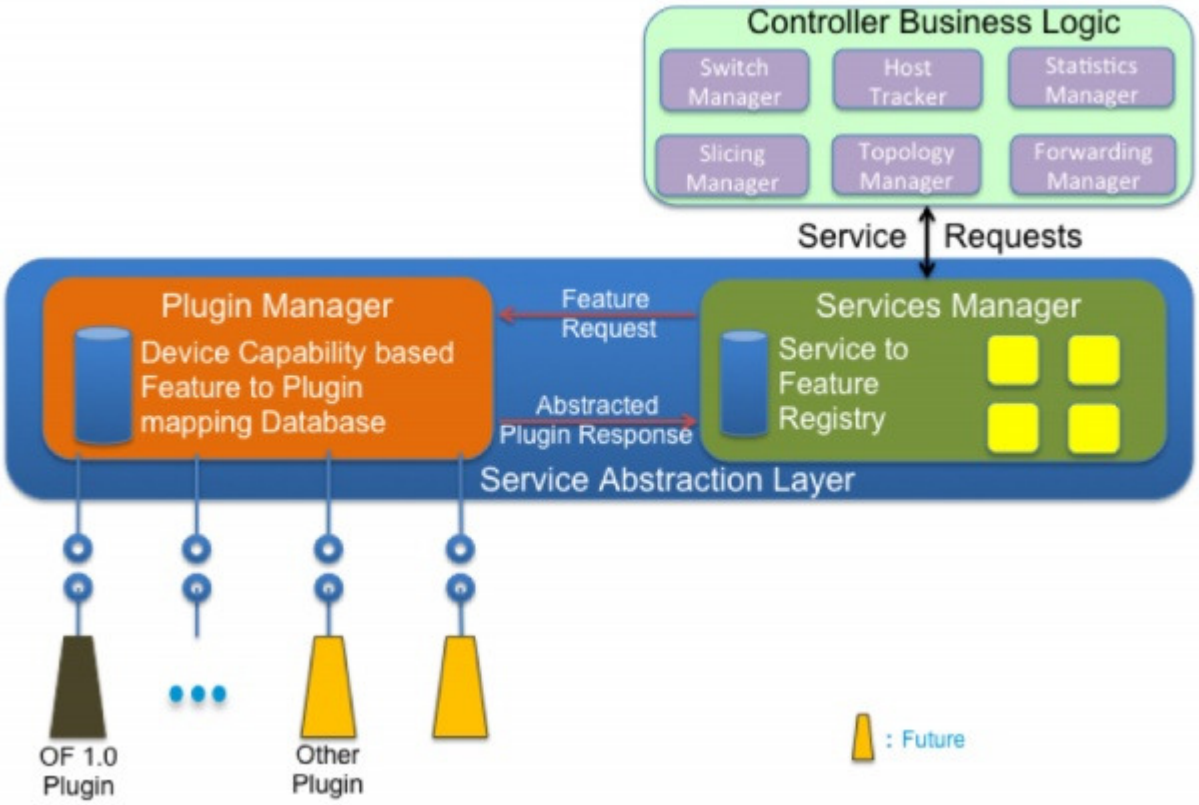


# Key repositories of ODL

- **Controller** : Core Controller functionality including MD-SAL and Base NSF (Network Service Function).
- **OpenFlowPlugin** : Southbound Plugin for Openflow Protocol communication.
- **OpenFlowjava** : Library for Serialization/De-Serialization of OpenFlow messages.
- **YangTools** : Handles Code generation parsing Yang Models and RestConf.
- **Integration Repo** : Integrates all the ODL Projects as a deliverable.

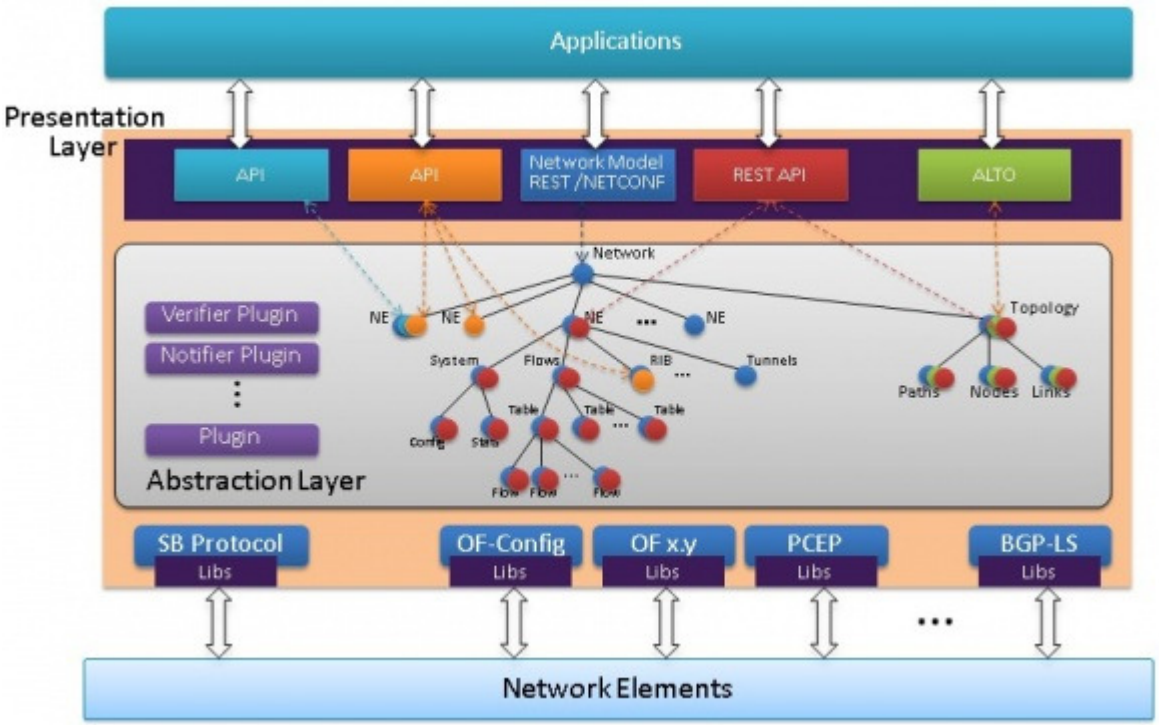
# SAL

# Service Abstraction Layer (SAL)




OPEN DAYLIGHT FORUM INDIA 2015

# Evolution of SAL




OPEN DAYLIGHT FORUM INDIA 2015

# MD-SAL

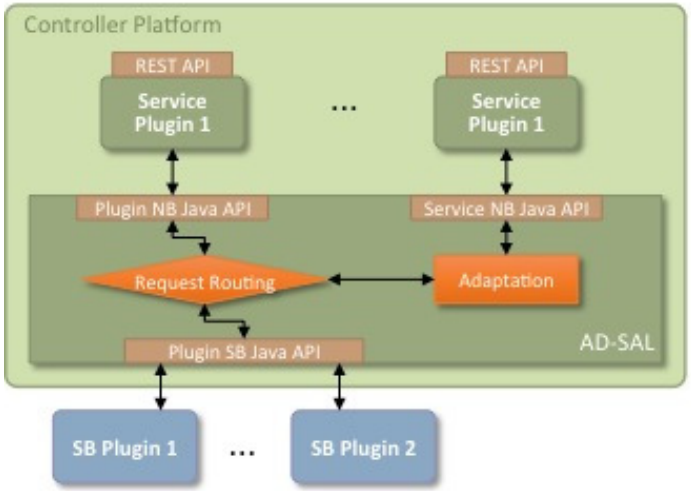
# MD-SAL

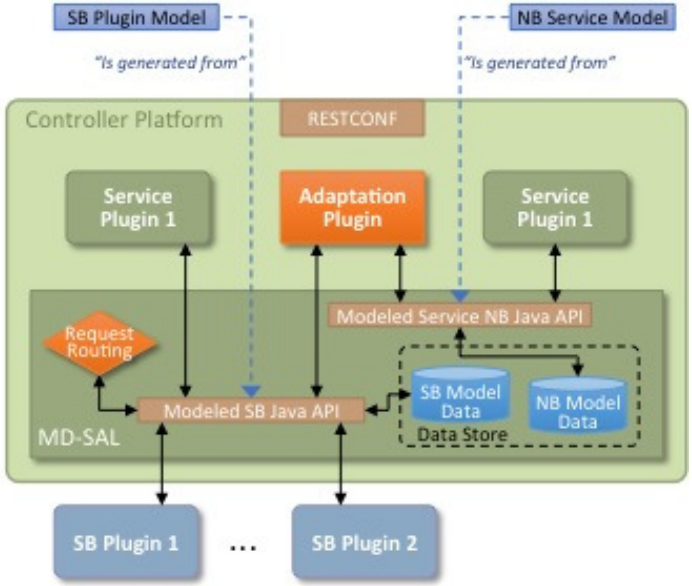
- Model-Driven SAL (MD-SAL) is a set of infrastructure services aimed at providing common and generic support to application and plugin developers.

- MD-SAL currently provides infrastructure services for:
  - Data Store
  - RPC / Service routing
  - Notification subscription and publish services

# AD-SAL to MD-SAL

See how to convey that there are two SALs , without getting into more details of either AD-SAL or MD-SAL at this point



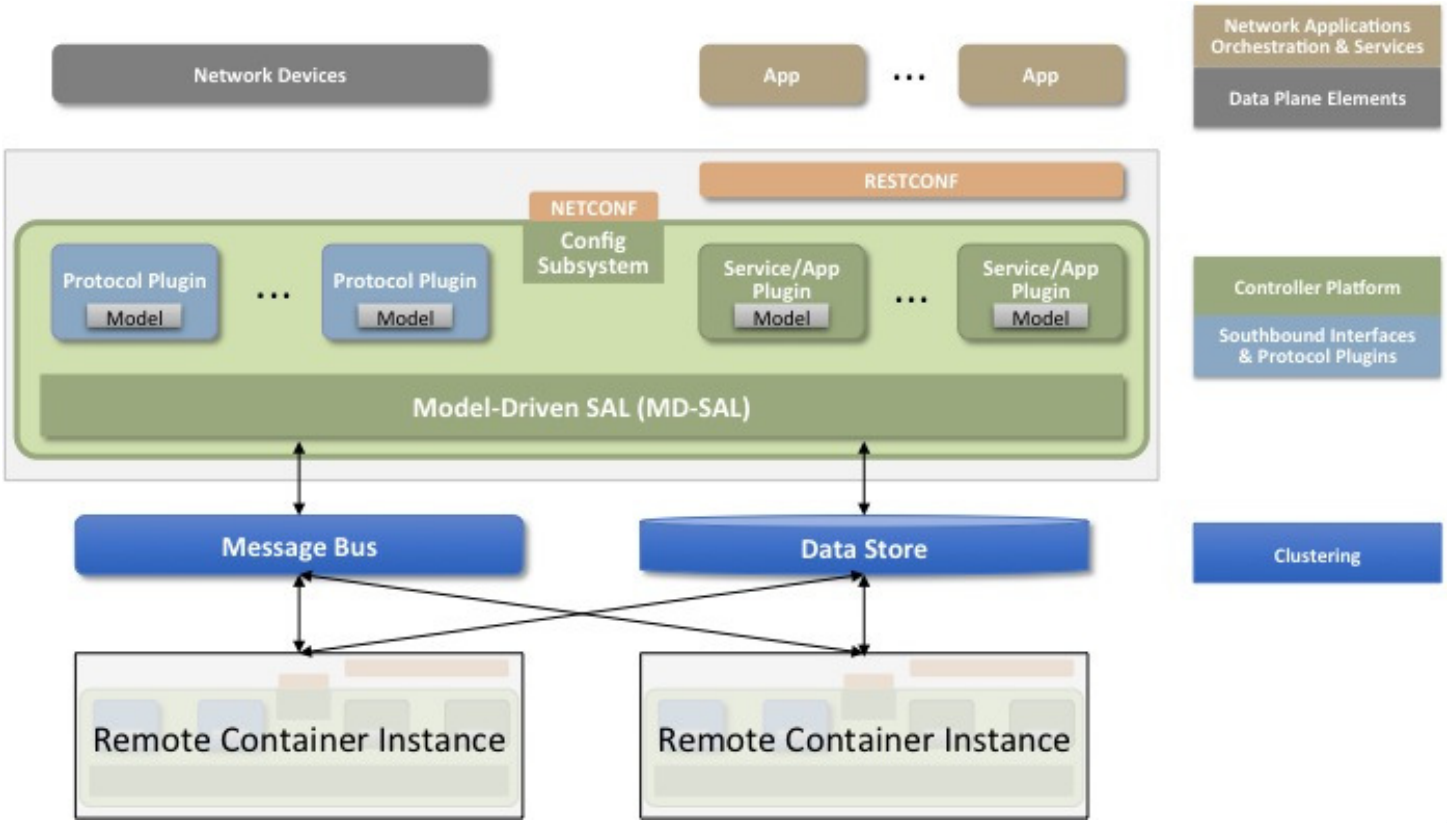




AD-SAL key services:
- Request routing
- Service Adaptation

MD-SAL key services:
- Request (RPC) and notification routing
- Data Storage

OPEN DAYLIGHT
FORUM INDIA 2015

# MD-SAL – programmer's view



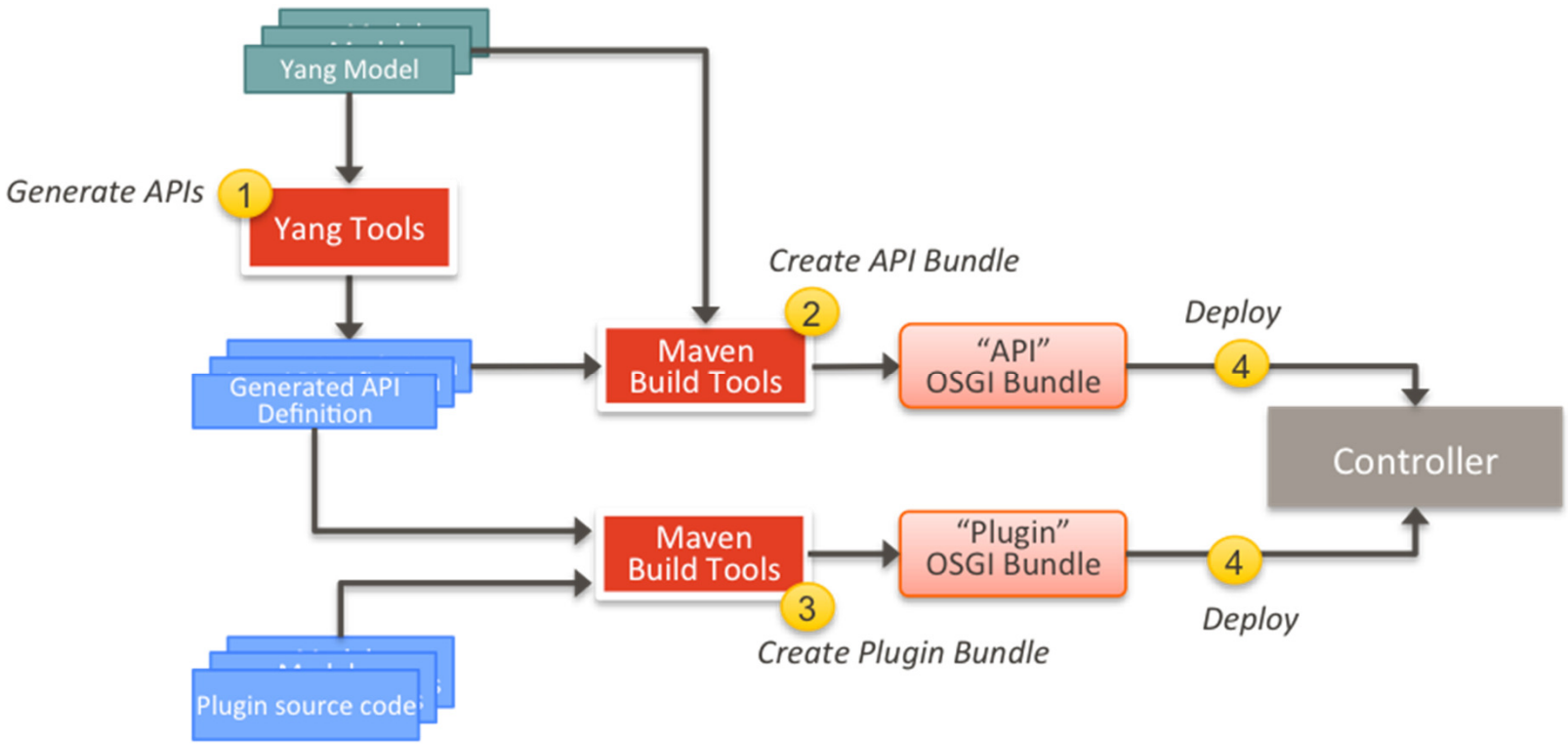

OPEN DAYLIGHT FORUM INDIA 2015

# MD-SAL plugin

# MD-SAL Plugin Types

- **Southbound Protocol Plugin**
- **Manager-type Application**
- **Protocol Library**
- **Connector Plugin**

# Plugin Development Process



# Designing a plugin

- During the design phase, the plugin designer
  - decides which data will be consumed by the plugin and
  - imports the SAL APIs generated from the models of the API provider.

- The designer decides which data will be provided by the plugin  & and how

- The designer designs the data model for the provided data. The data model is then used to generate the SAL APIs for the model.

# Implementing a plugin

- The implementations for the generated consumer and provider APIs, along with other plugin features and functionality, are developed.
- The resulting code is packaged in a “plugin” OSGI bundle. Note that a developer may package the code of a subsystem in multiple plugins or applications that may communicate with each other through the SAL.
- The generated APIs and a set of helper classes are also built and packaged in an “API” OSGI bundle.

# Data modelling & YANG

# Data modelling

- In order to describe the structure of data provided by controller components, a domain-specific modeling language to model services and data abstractions is to be used.

- Such language would allow
  - Modeling the structure of XML data and functionality provided by controller components.
  - Defining semantic elements and their relationships.
  - Modelling all the components as a single system.

# YANG Modelling : Introduction

- YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol.

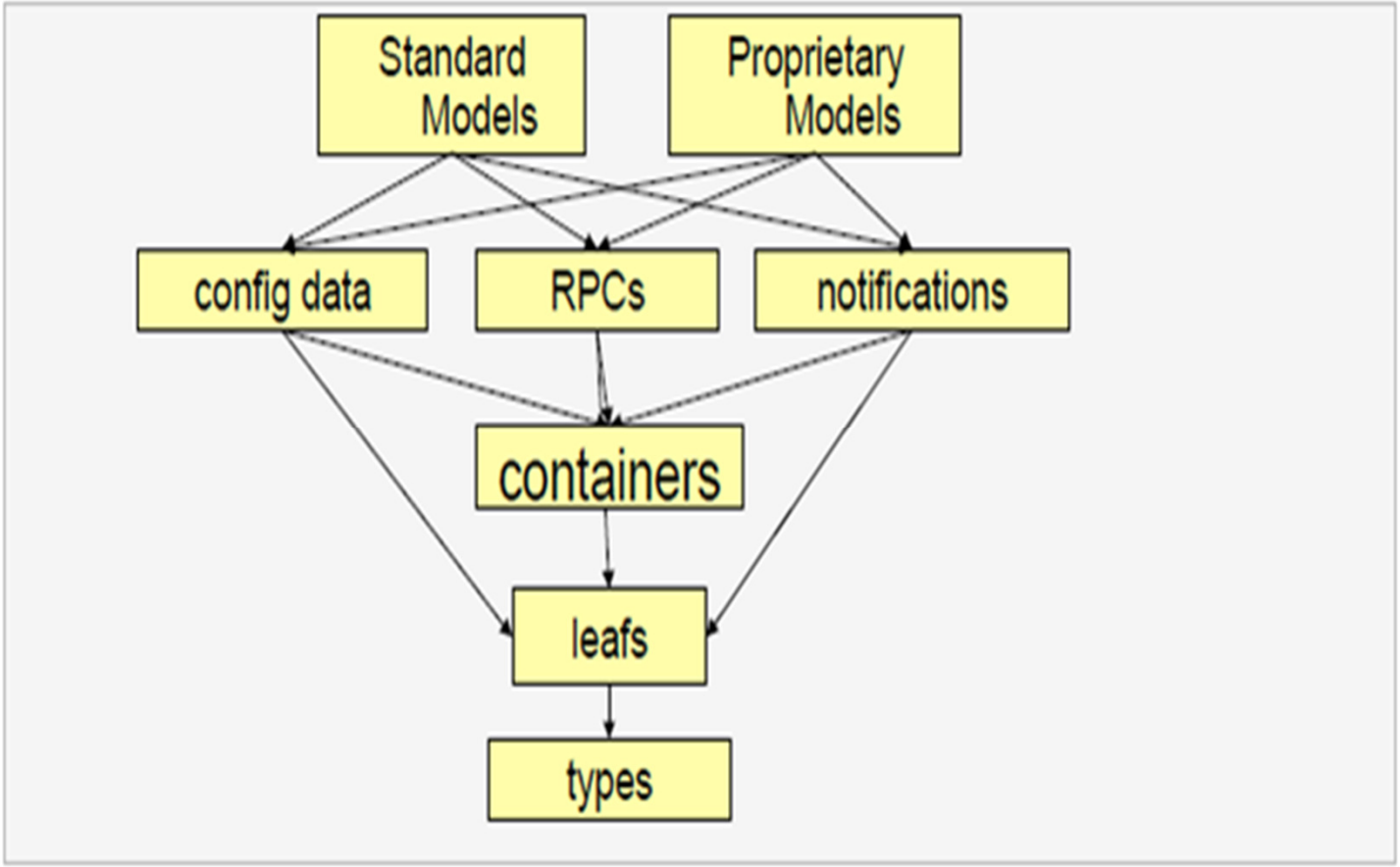
- YANG models the hierarchical organization of data as a tree in which each node has a name, and either a value or a set of child nodes.

- The XML nature of YANG data model presents an opportunity for self-describing data

# Contd

- Controller components and applications using the controller's northbound APIs can consume this in a raw format, along with the data's schema.
- Utilizing a schema language simplifies development of controller components and applications.
- Developer of a module that provides some functionality (a service, data, functions/procedure) can define a schema and thus create simpler APIs for the provided functionality, and thereby lower the risk of incorrect interpretation of data structures exposed through the SAL.

# YANG - concepts




OPEN DAYLIGHT FORUM INDIA 2015

# Dynamic evaluation of YANG at run time

- When ODL loads a bundle it searches that bundle (JAR) for files that end in “*.yang”.
- For each YANG file found it processes that file and builds several things dynamically including the structures to expose that as a RESTCONF compliant YANG interface.
- It also uses JavaAssist to dynamically create and compile classes that are used to marshal the YANG objects between POJOs and a XML/JSON representation.
- Lastly it maintains information based on the yang to translate between what ODL calls a binding aware class (POJO) and RPC (Java method) and binding independent implementations of the RPCs (such as you might find implemented on a router that is based on NETCONF).

# Karaf

# Karaf

- An OSGi-based runtime container
- Allows dynamic deployment of application bundles.
- Allows runtime installation, uninstallation of the deployed bundles without reboot.
- Application generally deployed as a feature.
- Feature – a set of bundles with an optional set of configuration files.

# Feature

Features are defined in feature xml descriptor as

```
<features
xmlns="http://karaf.apache.org/xmlns/features/v1.2.0">
<feature name="feature1" version="1.0.0">
<bundle>...</bundle>
<configfile finalname="“config filename” </configfile>
</feature>
</features>
```

# Build

# Maven

- A standard way to build the projects
- A clear definition of what the project consisted of
- An easy way to publish project information and a way to share JARs across several projects
- Inherently project-centric : everything revolves around the notion of a project.
- Maven essentially provides a way to help with managing:
  - Builds
  - Documentation
  - Reporting
  - Dependencies
  - SCMs
  - Releases
  - Distribution

# Project Object Model

- POM is the basic unit of work in Maven.
- POM contains every important piece of information about project
- Is essentially a one-stop-shopping for finding anything related to your project.
- Minimum requirement for a POM are the following:
  - project root
  - modelVersion - should be set to 4.0.0
  - groupId - the id of the project's group.
  - artifactId - the id of the artifact (project)
  - version - the version of the artifact under the specified group

# Maven Phases & Archetypes

- Maven Phases
  - Validate
  - Compile
  - Test
  - Package
  - Integration-test
  - Verify
  - Install
  - Deploy
  - Clean
  - Site

- Maven Archetype
  - A model from which all other things of the same kind are made
  - Archetypes are “templates” of applications that can be used to generate
  - ODL Maven archetype : odl-model-project

# Testing your code

# Testing your code

- JUNIT
  - White-box testing framework tied with the maven using jacoco surefire plugin.
  - Test code added in the same package name with suffix *Test.java
  - Test methods has to be written with public access and should not return any value.
  - Test is considered to be a failure when  exception is thrown by the test method.
    - *eg public void testMethod() throws Exception*
  - Assert class and verify method can be used for the unit test verification

# JUNIT

| Annotations | Description |
|---|---|
| @Test | Specifies the method as the test method |
| @Test (expected = Exception.class) | Fails if test method doesn't throw the given exception |
| @Before | Used to initialize the test variables; executes before each test. |
| @After | Executed after each test to cleanup the test environment |
| @BeforeClass | Static method executed once, before the start of all tests<br>Performs common one-time initialization for the test environment. |
| @AfterClass | Static method executed once, after the finish of all tests<br>Performs clean-up of the test environment. |

# Mockito

- A mocking framework
- Creates dummy objects(mock objects) to be used in the testing.
- Avoids creation of real test objects
- Useful in testing methods involving inter-bundle communication.

# Working with ODL Code

# Working with ODL Code

- Steps to follow when working with the ODL code base
  - Set up the Gerrit account
  - Pull the Code from ODL repository
  - Run the ODL Helium

# Build - Precursor

- *Maven is a Java tool, so you must have* Java *installed*
- Download and install maven
  - *OpenDaylight has its Nexus repository*
  - *.m2/settings.xml controls how code is pulled & compiled.*
  - *https://wiki.opendaylight.org/view/GettingStarted:Development_Environment_Setup*
  - Optional: Increase the amount of RAM maven can use
    - export MAVEN_OPTS='-Xmx1048m -XX:MaxPermSize=512m'

# Thank You Note

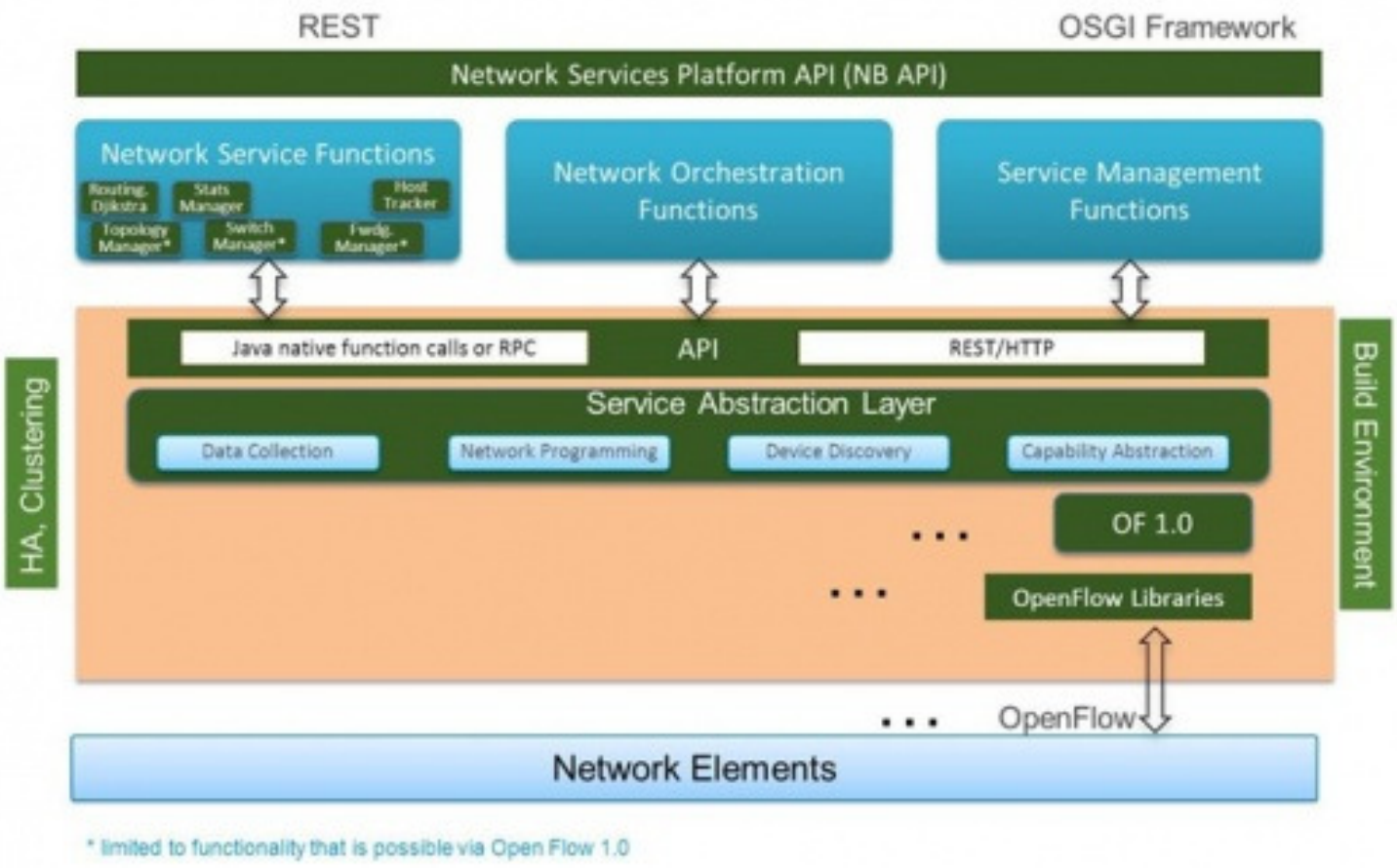
Appreciate the ODL Community, wiki, previous ODL summit speakers and enthusiasts for wealth of information they have created on ODL

OPEN DAYLIGHT
FORUM INDIA 2015

# Backup Slides

# ODL Backup Slides

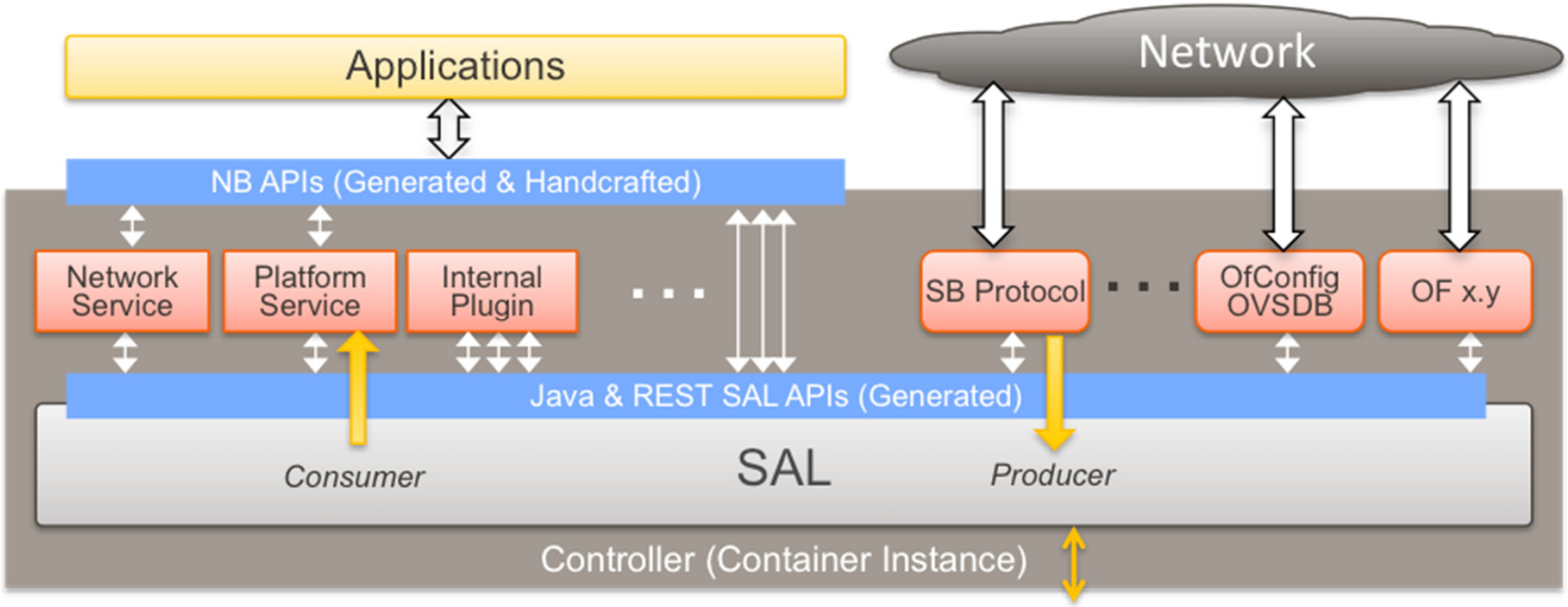
# ODL Framework overview




OPEN DAYLIGHT FORUM INDIA 2015

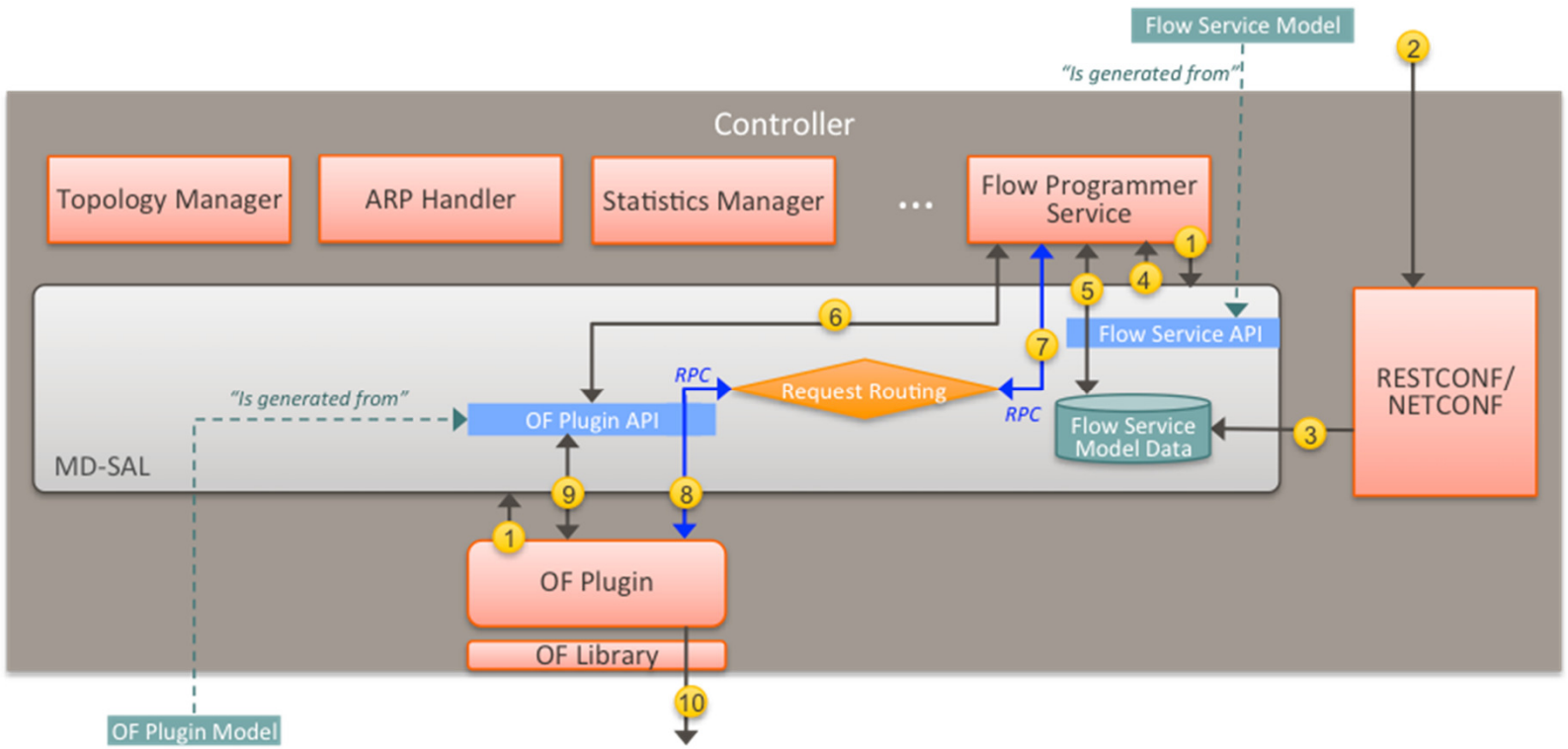
# MD-SAL / Flow Back-up Slides

# SAL & plugins



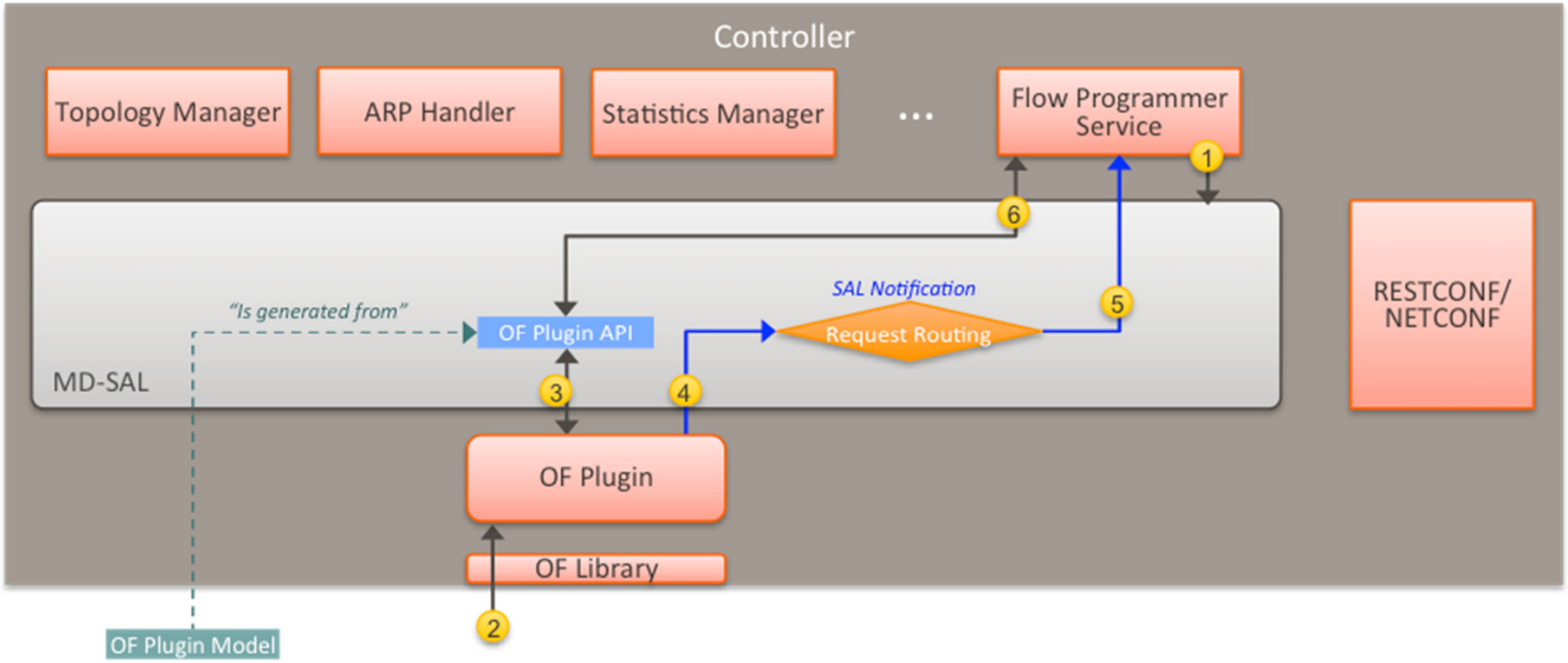

OPEN DAYLIGHT
FORUM INDIA 2015

# Add Flow via NB Rest API




OPEN DAYLIGHT FORUM INDIA 2015

# Delete flow from switch




OPEN DAYLIGHT FORUM INDIA 2015

# How MD-SAL identifies SB Plugin for flow provisioning

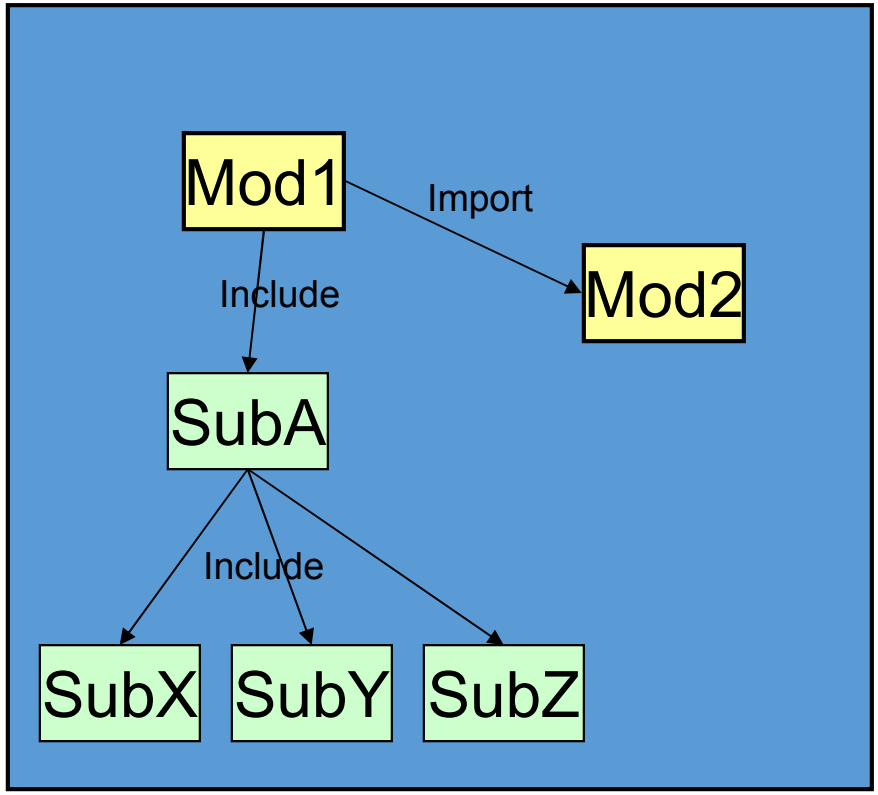
- In SB plugin, extend SalFlowService (an RpcService that provides APIs to add, delete and update flows ) to provide an implementation of addFlow, removeFlow and updateFlow.
  - Use *addRoutedRpcImplementation* method here.
- For flow service, MD-SAL identifies the southbound plugin using node. So southbound plugin needs to tell RoutedRpcRegistration for flow service that it has the provider
- Then register this instance identifier with the RoutedRpcRegistration (a BindingAwareBroker interface) for flow service to beprocessed by a given RpcService.
- Whenever a new device is connected using southbound plugin, create an instance identifier and register it with flow registration.
- Now, whenever MD-SAL gets a flow provisioning request , it routes it to southbound plugin.

# YANG additional slides

# YANG – An Introduction

- Models semantics and data organization
  - Models configuration and state data manipulated by the Network Configuration Protocol.
  - Syntax falls out of semantics
- Ability to model RPCs, and notifications

# Modules and submodules



```
module acme-module {
    namespace "http://acme.example.com/module";
    prefix acme;

    import "yang-types" {
        prefix yang;
    }
    include "acme-system";

    organization "ACME Inc.";
    contact joe@acme.example.com;
    description "The module for entities
                 implementing the ACME products";

    revision 2007-06-09 {
        description "Initial revision.";
    }
    …
}
```

# The "leaf" Statement

- A leaf has
  - one value
  - no children
  - one instance

```
YANG Example:

leaf host-name {
    type string;
    mandatory true;
    config true;
    description "Hostname for this system";
}
```

```
XML Encoding:

<host-name>my.example.com</host-name>
```

# The "leaf-list" Statement

- A leaf-list has
  - one value
  - no children
  - multiple instances

```
YANG Example:

leaf-list domain-search {
    type string;
    ordered-by user;
    description "List of domain names to search";
}
```

```
XML Encoding:

<domain-search>high.example.com</domain-search>
<domain-search>low.example.com</domain-search>
<domain-search>everywhere.example.com</domain-search>
```

# The "container" Statement

```
YANG Example:

container system {
    container services {
        container ssh {
            presence "Enables SSH";
            description "SSH service specific configuration";
             // more leafs, containers and stuff here...
        }
    }
}
```

- A container has
  - no value
  - holds related children
  - one instance

```
XML Encoding:

<system>
  <services>
     <ssh/>
  </services>
</system>
```

May have specific meaning (presence)

Or may simply contain other nodes

# The "list" Statement

```
YANG Example:

list user {
    key name;
    leaf name {
        type string;
    }
    leaf uid {
        type uint32;
    }
    leaf full-name {
        type string;
    }
    leaf class {
        type string;
        default viewer;
    }
}
```

```
XML Encoding:

<user>
  <name>glocks</name>
  <full-name>Goldie</full-name>
  <class>intruder</class>
</user>
<user>
  <name>snowey</name>
  <full-name>Snow</full-name>
  <class>free-loader</class>
</user>
<user>
  <name>rzull</name>
  <full-name>Repun</full-name>
</user>
```

- A list is
  - uniquely identified by key(s)
  - holds related children
  - no value
  - multiple instances

# Built-in types

| Category | Types |
|---|---|
| Integral | {,u}int{8,16,32,64} |
| String | string, enumeration, boolean |
| Binary Data | binary |
| Bit fields | bits |
| References | instance-identifier, keyref |
| Other | empty |

# Derived types

```
YANG Example:

typedef percent {
    type uint16 {
        range "0 .. 100";
    }
    description "Percentage";
}

leaf completed {
    type percent;
}
```

```
XML Encoding:

<completed>20</completed>
```

- Constraints
  - range
  - length
  - pattern
    - regex
- A modules may use types imported from other modules

# The "union" type

```
YANG Example:

leaf limit {
    description "Number to allow";
    type union {
        type uint16 {
            range "0 .. 100";
        }
        type enumeration {
            enum none {
                description "No limit";
            }
        }
    }
}
```

- Allows a leaf to contain a superset of types

```
XML Encoding:

<limit>none</limit>
```

```
XML Encoding:

<limit>20</limit>
```

# The "rpc" Statement

```
rpc activate-software-image {
   input {
      leaf image-name {
          type string;
      }
   }
   output {
      leaf status {
          type string;
      }
   }
}
```

- Defines RPC
  - method names
  - input parameters
  - output parameters

```
<rpc xmlns="urn:mumble">
  <activate-software-image>
   <image-name>image.tgz</image-name>
  </activate-software-image>
</rpc>
```

# The "notification" Statement

- Defines notification
  - Name
  - Content

```
YANG Example:

notification link-failure {
    description "A link failure has been detected";
    leaf if-index {
        type int32 { range "1 .. max"; }
    }
    leaf if-name {
         type keyref {
            path "/interfaces/interface/name";
        }
    }
}
```

# The "augment" Statement

```
YANG Example:

augment system/login/user {
    leaf expire {
        type yang:date-and-time;
    }
}
```

- Extends data model
  - Current or imported modules
- Inserts nodes
  - Into an existing hierarchy
  - Nodes appear in current module's namespace
  - Original (augmented) module is unchanged

```
XML Encoding:

<user>
  <name>alicew</name>
  <class>drop-out</class>
  <other:expire>2112-04-01T12:00:00</other:expire>
</user>
```

# Semantic Differentiators

- Notice that YANG is modeling the semantics and data organization
  - Not just the syntax

| Statement | Purpose |
|---|---|
| unique | Ensure unique values within list siblings |
| keyref | Ensure referential integrity |
| config | Indicate if a node is config data or not |
| default | Supply default value for leafs |
| error-app-tag | Define the tag used when constraint fails |
| error-message | Define the message used .... |
| mandatory | Node must exist in valid config datastore |

OPEN
DAYLIGHT
FORUM INDIA 2015