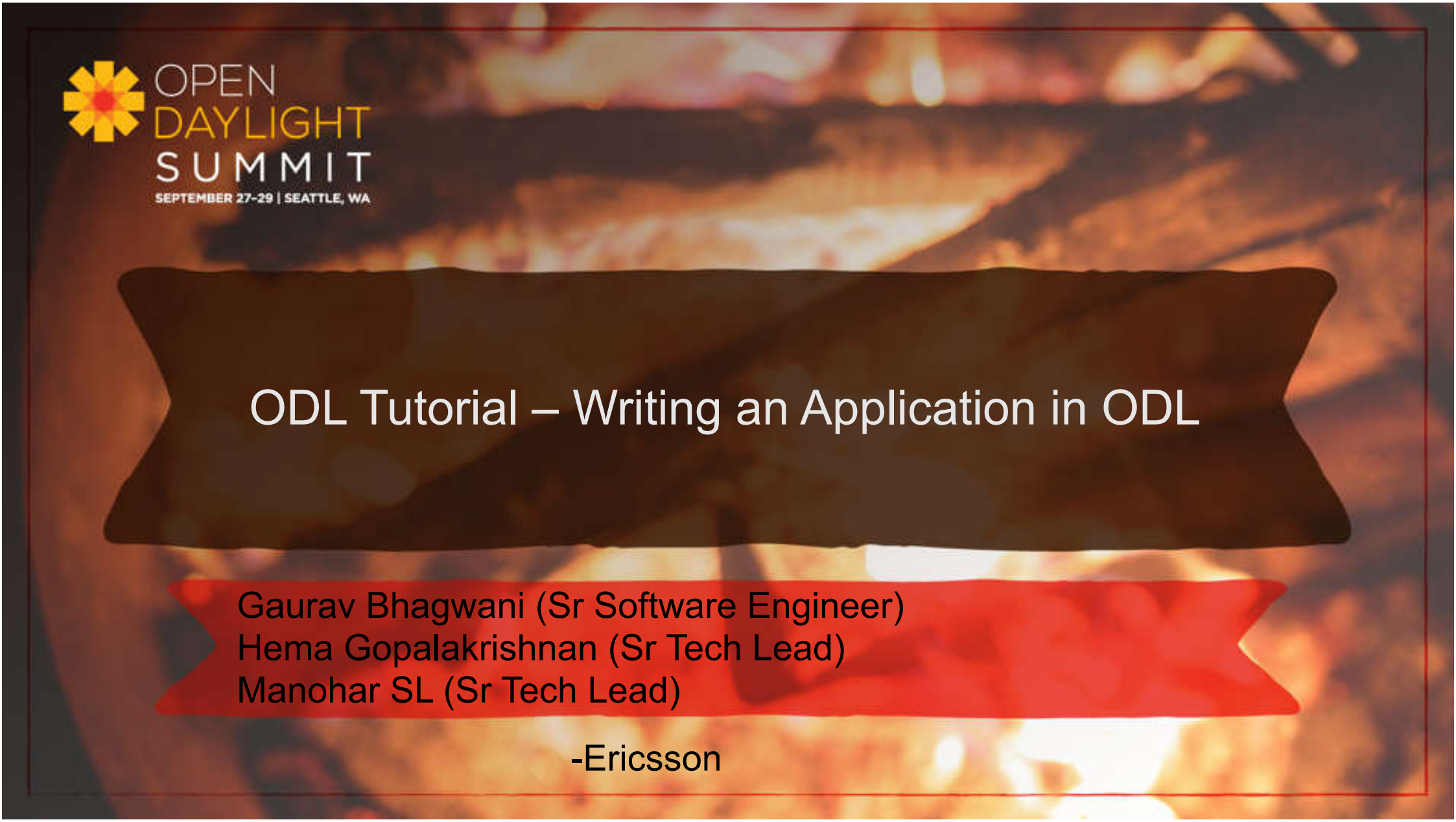


ODL Tutorial – Writing an Application in ODL

Gaurav Bhagwani (Sr Software Engineer)
Hema Gopalakrishnan (Sr Tech Lead)
Manohar SL (Sr Tech Lead)

-Ericsson

Environment Setup

- Get a USB Key
- Copy the contents to the laptop
- Install Virtual Box for your laptop
- Unzip ODL-Summit-2016-Tutorial-2016.zip
- Import the OVA into Virtual Box or VMWare Fusion
- Boot up the Virtual Machine and Login
- Login User/Password: odl-developer/odl-developer

Goals

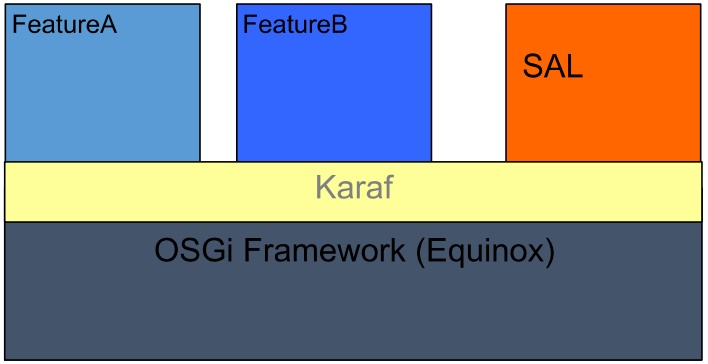
This hands-on tutorial will walk you through writing a sample OpenDaylight application and test the data path: In this journey, we will

- Explore the different technologies used in ODL such as OSGI, Karaf, Maven, Yang Modelling, Blueprint container, listing some useful commands / options.
- Discuss the Opendaylight Architecture
- Write a sample app from Maven arche type
- Register with DataStore for any data change.
- Use the REST APIs to push configuration data to the sample app
- Induce dependency on other features
- Install flows on the switch using Opendaylight OF plugin
- Connect switches and test the datapath

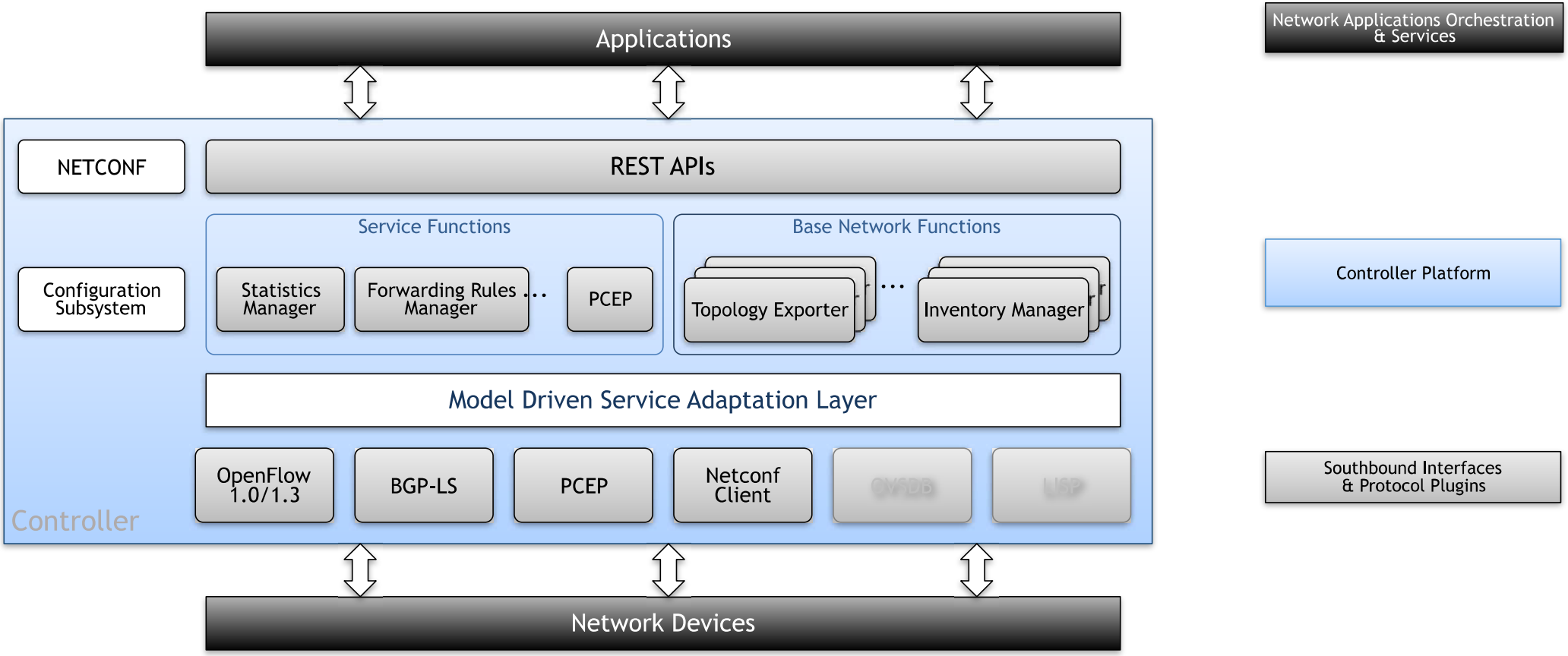
- Along the way we will discuss some best practices.

Technologies used in ODL

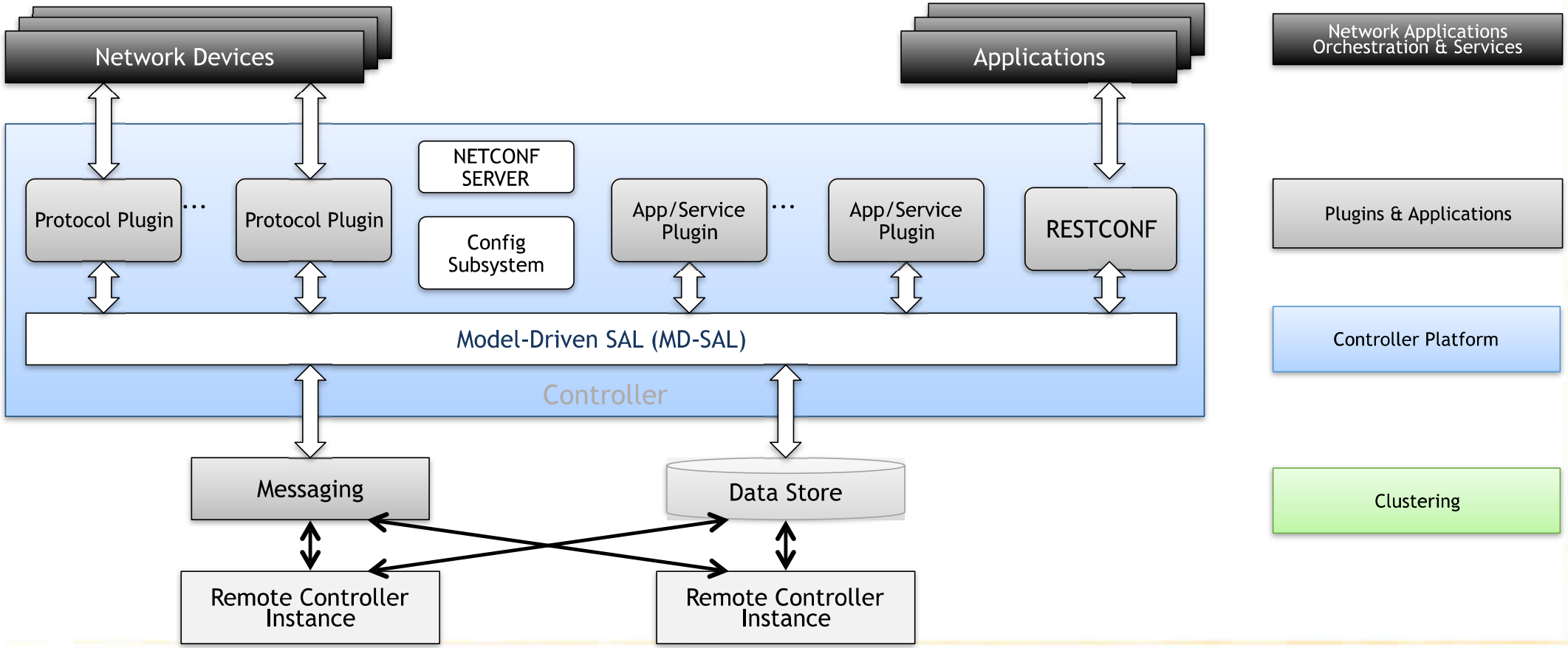
- Language used in ODL – Java.
- Language used for Modelling - YANG
- Build System used in ODL Maven
- Basic Platform of ODL – Karaf, which is powered by OSGi
  - Hot Deployment
  - Dynamic loading modules/bundles
  - You can connect remotely to Karaf console using SSH
  - Dynamic configuration – All karaf configuration files are in /etc
- Blueprint Container for dependency injection across bundles that run in an OSGi framework.



OpenDaylight: SDN Controller Architecture



OpenDaylight: Software Architecture



ODL Platform Overview

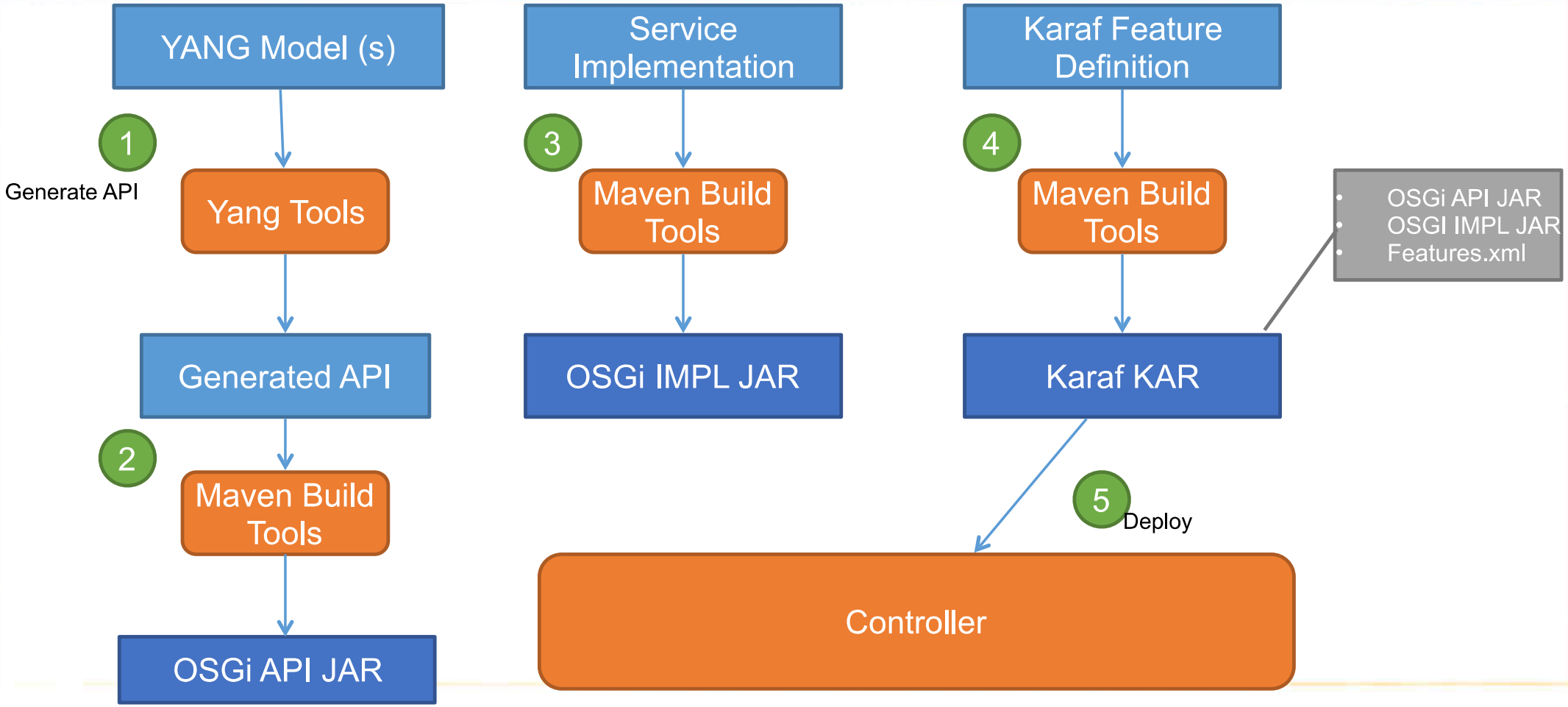
- **Micro Services Architecture**
  - ODL employs a model-driven approach to describe the network, the functions to be performed on it and the resulting state
  - In MDSAL, any app or function can be bundled into a service that is then then loaded into the controller.
  - Only install the protocols and services you need
  - Fine-grained services to be created then combined together to solve more complex problems.

- **Multiprotocol Support**
  - ODL platform supports OpenFlow , OpenFlow extensions, NETCONF, BGP/PCEP. Additionally, ODL interfaces with OpenStack and Open vSwitch

Tools for development

- A text editor, preferably an IDE like IntelliJ IDEA or Eclipse
- YANG as a Modeling language (based on RFC 6020)
- Java 1.8 as a Programming language
- Maven 3.2.5 as a Build tool
- OSGi – for building modular systems
- Karaf – for deploying and managing OSGi bundles
- Blueprint – for dependency injection

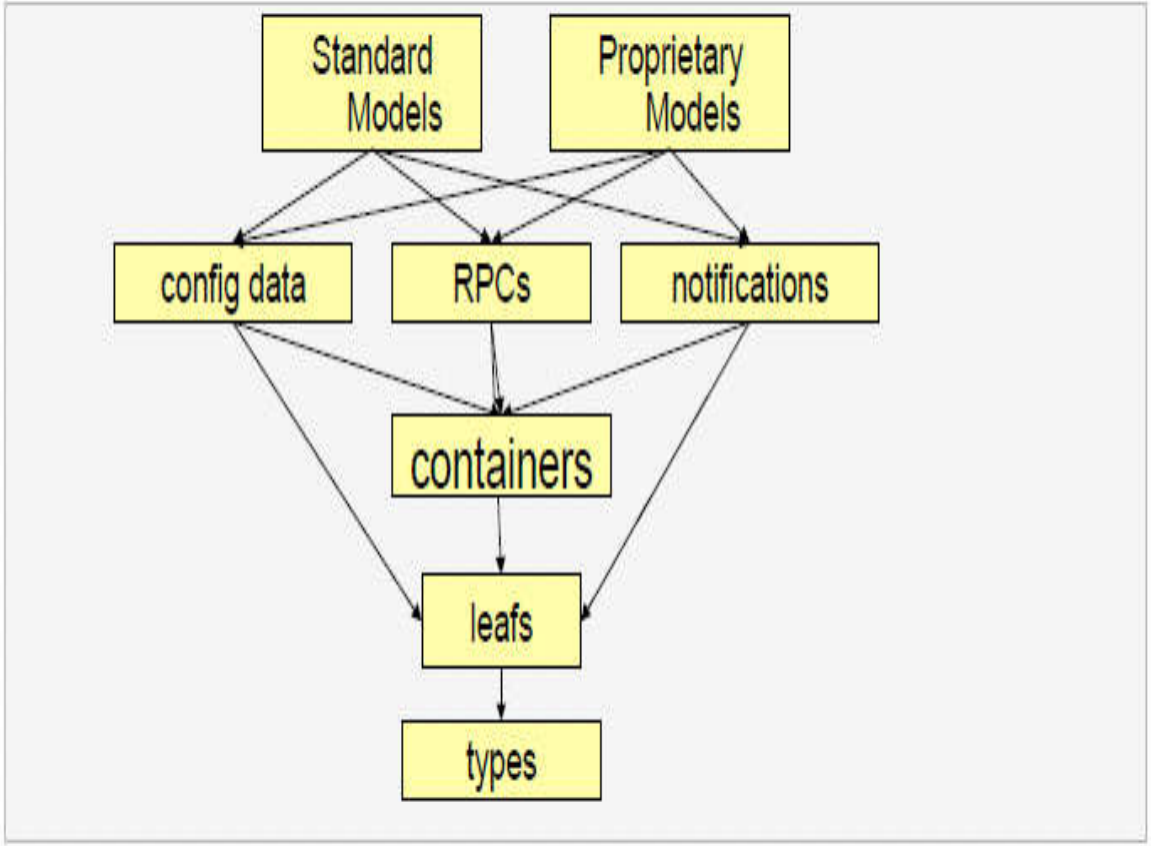
The Service Development Process



Yang Modelling – Basic Concepts

- YANG is a data modeling language used to model configuration and state data manipulated.

- 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.



Data Modeling

Some YANG constructs and the corresponding Java generated code

Yang Construct	Mapped Java Code
Container	JAVA interface which extends interfaces DataObject, Augmentable<container_interface>
Leaf	The leaf is mapped to getter method of superior element with return type equal to typesubstatement value.
Leaf-List	The leaf-list is mapped to getter method of superior element with return type equal to Listof type substatement value.
List	YANG list element is mapped to JAVA interfac
Choice and case	Choice element is mapped similarly as list element. Case substatements are mapped to the JAVA interfaces which extend mentioned marker interface.

Data Modeling (Contd.)

Some YANG constructs and the corresponding Java generated code

Yang Construct	Mapped Java Code
Grouping Uses used in some element	Grouping is mapped to JAVA interface. Uses are mapped as extension of interface for this element with the interface which represents grouping.
RPC with Input and Output	RPC is mapped to an method of a Java Class with input and output parameters mapped to the corres. Input and Output
Augment	Augment is mapped to the JAVA interface. The interface starts with the same name as the name of augmented interface.

YangTools Advantages

- Generates Java code from Yang
  - This Data Transfer Objects (DTOs) is
    - **Immutable**: to avoid thread contention
    - **Strongly typed:** reduce coding errors
    - **Consistent**: reduce learning curve
    - **Improvable** – generation can be improved and all DTOs get those improvements immediately system wide
- Provides 'Codecs' to convert
  - Generated Java classes to DOM
  - DOM to various formats
    - XML
    - JSON
- 'Codecs' provide automatic bindings
  - RESTCONF (XML and JSON)
  - NetConf

Karaf

Apache Karaf is a small OSGi based runtime which provides a lightweight container onto which various components and applications can be deployed.

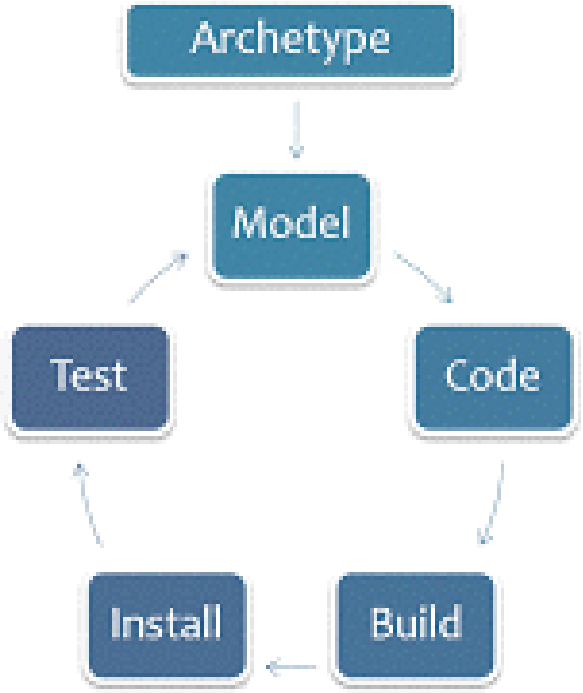
Some of the key advantages of Apache Karaf are listed below:

- **Hot deployment**: Karaf supports hot deployment of OSGi bundles by monitoring jar files inside the [home]/deploy directory. Each time a jar is copied in this folder, it will be installed inside the runtime. You can then update or delete it and changes will be handled automatically. In addition, the Karaf also supports exploded bundles and custom deployers (blueprint and spring ones are included by default).
- **Dynamic configuration**: Services are usually configured through the ConfigurationAdmin OSGi service. Such configuration can be defined in Karaf using property files inside the [home]/etc directory. These configurations are monitored and changes on the properties files will be propagated to the services.
- **Logging System**: using a centralized logging back end supported by Log4J, Karaf supports a number of different APIs (JDK 1.4, JCL, SLF4J, Avalon, Tomcat, OSGi)
- **Provisioning**: Provisioning of libraries or applications can be done through a number of different ways, by which they will be downloaded locally, installed and started.
- **Extensible Shell console**: Karaf features a nice text console where you can manage the services, install new applications or libraries and manage their state. This shell is easily extensible by deploying new commands dynamically along with new features or applications.
- **Remote access**: use any SSH client to connect to Karaf and issue commands in the console
- **Security framework** based on JAAS

Blueprint

- Dependency injection frameworks support us in writing modular, flexible, testable and clean code
- The "Blueprint Container Specification" defines a dependency injection framework to build applications that run in an OSGi framework.
- ODL had the config subsystem as dependency injection framework.
- Config subsystem is a home grown solution that uses yang modelling language as a language for modelling the configuration, dependencies and state data for Modules.
- The config subsystem is an activation and configuration framework, similar to what Blueprint that allows run-time rewiring.
- Blueprint is more user friendly and is an alternate to config subsystem

STEPS FOR ODL APPLICATION DEVELOPMENT



ArcheType


```
mvn archetype:generate -
DarchetypeGroupId=org.opendaylight.controller \
-DarchetypeArtifactId=opendaylight-startup-archetype \
-DarchetypeVersion=1.3.0-SNAPSHOT \
-DarchetypeRepository=http://nexus.opendaylight.org/content/
repositories/opendaylight.snapshot/ \
-DarchetypeCatalog=http://nexus.opendaylight.org/content/repositories/
opendaylight.snapshot/archetype-catalog.xml
```

Archetype(Contd.)

You will be prompted for :-

- groupId: (enter your project groupId)
- artifactId: project artifact Id
- version: (version of your project)
- package: (accept default)
- classPrefix:usually derived from artifact Id
- copyright: company name
- copyrightYear: 2016

Archetype Example for Sample App

- groupId: (org.opendaylight.sampleapp)
- artifactId: sampleapp
- version: (0.1.0-SNAPSHOT)
- package: (accept default) (org.opendaylight.sampleapp)
- classPrefix:Sampleapp
- copyright: Opendaylight
- copyrightYear: 2016

Model –Yang Model Example


```
grouping acl-ipv4-header-fields {
    description "fields in IPv4 header";

    leaf destination-ipv4-address {
        type inet:ipv4-prefix;
    }

    leaf source-ipv4-address {
        type inet:ipv4-prefix;
    }

}
```

```
container actions {
  description "Define action criteria";
  choice packet-handling {
    default deny;

    description "Packet handling action.";
    case deny {
      leaf deny {
        type empty;
        description "Deny action.";
      }
    }
    case permit {
      leaf permit {
        type empty;
        description "Permit action.";
      }
    }
  }
}
```

Code

- We will add the Code in phases
  - Phase 0
    - Just a log message in SampleappProvider.java
  - Phase 1
    - Define a Yang Model and access the datastore using REST API
  - Phase 2
    - Register a ClusteredDataTreeChangeListeners to listen for Changes
    - Read the incoming data and log a message in the callback
  - Phase 3
    - Install flows in the switch
    - Inject the necessary dependencies such as dataBroker using Blueprint
    - Wire up the necessary feature dependencies such as OpenFlowPlugin using feature.xml

Build Parameters

Maven is the build system used.

Copy the settings.xml file

cp -n ~/.m2/settings.xml{,.orig} ; \wget -q -O - https://
raw.githubusercontent.com/opendaylight/odlparent/stable/boron/

Use the mvn command

Mvn clean install

Options

-o

-nsu

-rf :<bundle_to_resume_from>

- mvn clean -Dmaven.repo.local= <path of m2> -gs <path of settings.xml> install

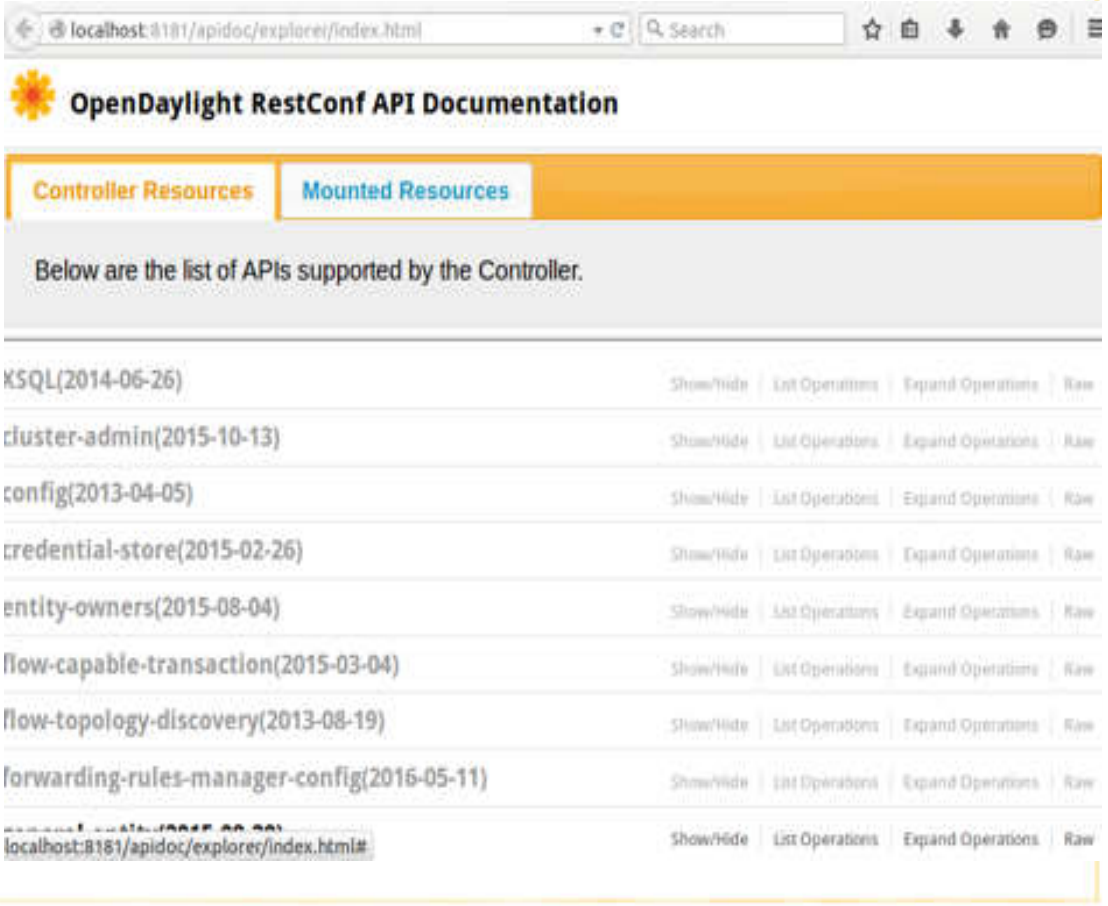
-U

Installation - Karaf

- Cd to <features>/karaf/target/assembly/bin
  - ./karaf (with clean option)
- On the karaf console
  - Opendaylight-user@root>feature:list
  - Opendaylight-user@root>feature:install <feature-name>
  - Opendaylight-user@root>bundle:list | grep <bundle-name>
  - Opendaylight-user@root>bundle:services <bundle-id>
  - Opendaylight-user@root>bundle:diag <bundle-id>

Test -- Rest APIs

- Opendaylight has significant REST API support
  - Restconf allows for checking config and operational state
    - Feature: install odl-restconf
  - WEB Browser lists the Northbound APIs that are auto generated
    - http://localhost:8181/apidoc/explorer/index.html



Exercise 1

Refer to the instructions in the ReadMe.txt under the directory /home/odl-developer/Tutorial in the VM

What is MD-SAL ?

Model-Driven SAL (MD-SAL) is the Kernel of OpenDaylight Controller
MD-SAL provides messaging and data storage functionality for based on user-defined (application developers) data and interface models.
This is achieved by using YANG as

-modelling language for interface
-data definition
-providing runtime for services

MD-SAL - Basic Modelling Concepts

- **Data Tree**

  All state-related data are modeled and represented as data tree,

  Operational Data Tree - Reported state of the network / system,
  - published by the providers using MD-SAL.
  - Not Persisted, Cannot be changed from REST

  Configuration Data Tree - Intended state of the system or network,
  - populated by consumers, which expresses their intention.
  - Persisted across Controller restarts Instance **Identifier -** Unique identifier of node / subtree in data tree

- **Instance Identifier** - Unique identifier of node / subtree in data tree,

MD-SAL: API Types

MD-SAL provides three API types:

1. Java generated APIs for consumers and producers.

2. DOM APIs: Mostly used by infrastructure components and useful for XML-driven plugin and application types.

3. REST APIs: Restconf that is available to consumer type applications and provides access to RPC and data stores

MD-SAL: RPC Service

- In MD-SAL terminology, the term *RPC* is used to define the input and output for a procedure (function) that is to be provided by a Provider, and adapted by the MD-SAL.

- In the context of the MD-SAL, there are two types of RPCs (RPC services):
  * Global: One service instance (implementation) per controller container or mount point.
  * Routed: Multiple service instances (implementations) per controller container or mount point.

MD-SAL: Global RPC Service

- There is only one instance of a Global Service per controller instance. Note that a controller instance can consist of a cluster of controller nodes.

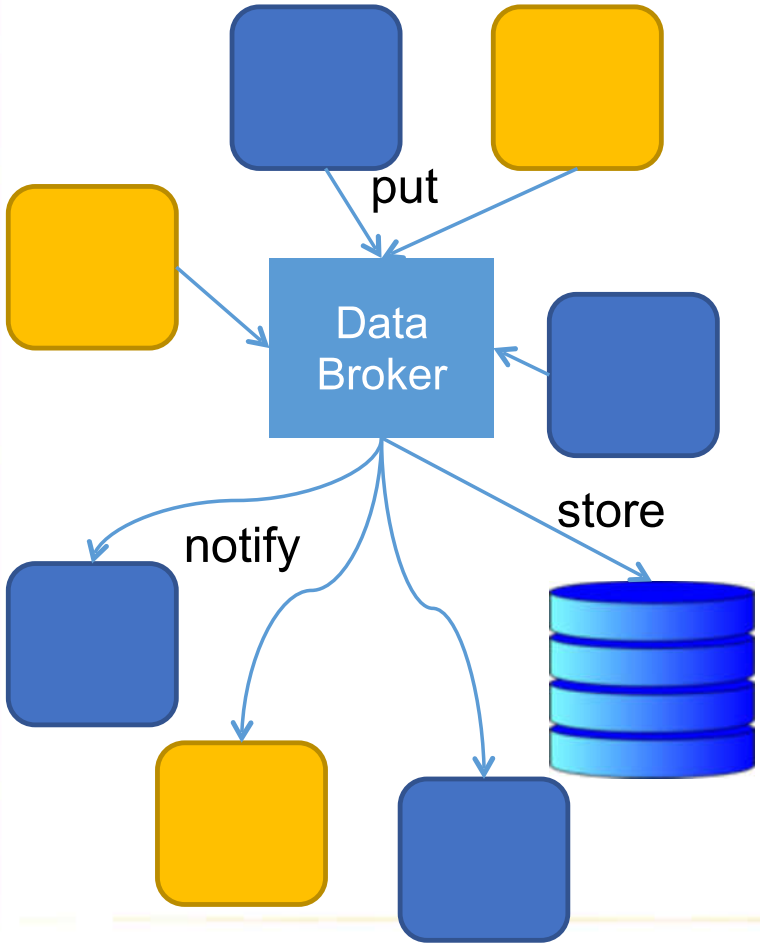
- Binding-Aware MD-SAL (sal-binding)
  - **RPC Type**: Identified by a generated RpcService class and a name of a method invoked on that interface.

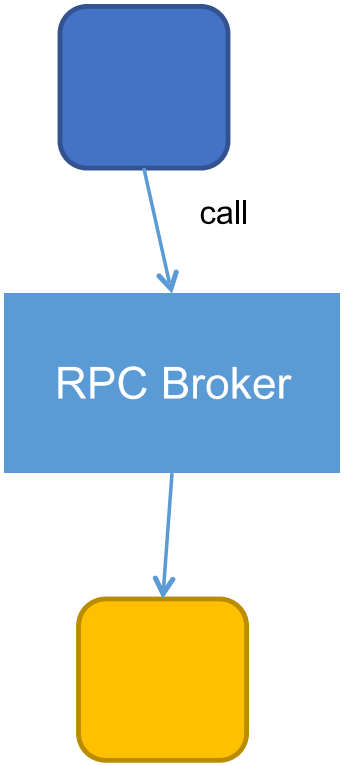
- Binding-Independent MD-SAL (sal-Dom)
  - **RPC Type**: Identified by a Qname.

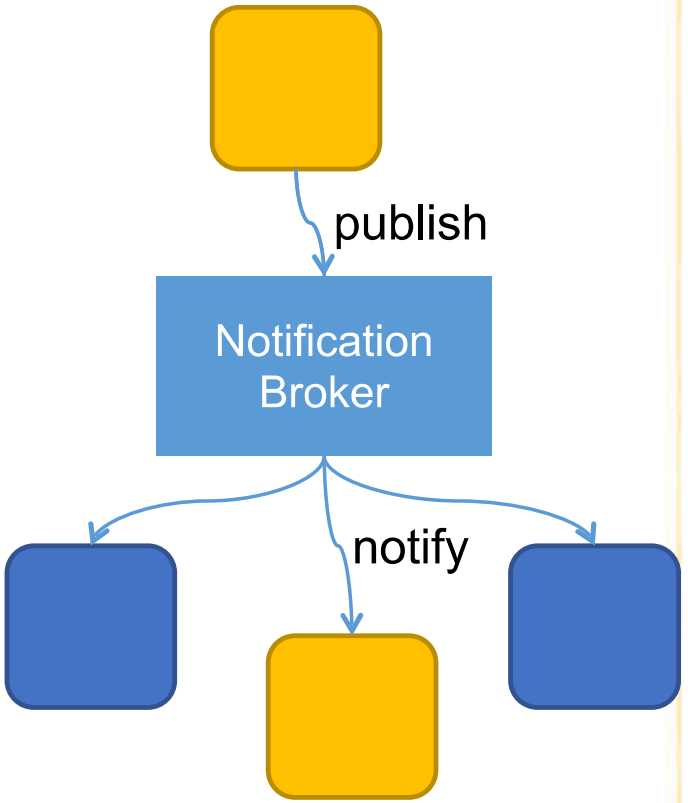
MD-SAL: Routed RPC Service

- There can be multiple instances (implementations) of a service per controller instance.
- Can be used for southbound plugins or for horizontal scaling (load-balancing) of northbound plugins (services).
- Routing is done based on the contents of a message, for example, *Node Reference*. The field in a message that is used for routing is specified in a YANG model by using the routing-reference statement from the yang-ext model.
  - Binding Aware MD-SAL (sal-binding)
  - RPC Type: Identified by an RpcService subclass and the name of the method invoked on that interface
  - Instance Identifier: In a data tree, identifies the element instance that will be used as the route target.
  - Binding Independent MD-SAL (sal-Dom)
  - RPC Type: Identified by a QName
  - Instance Identifier: In a data tree, identifies the element instance that will be used as the route target.

MD-SAL -Brokers



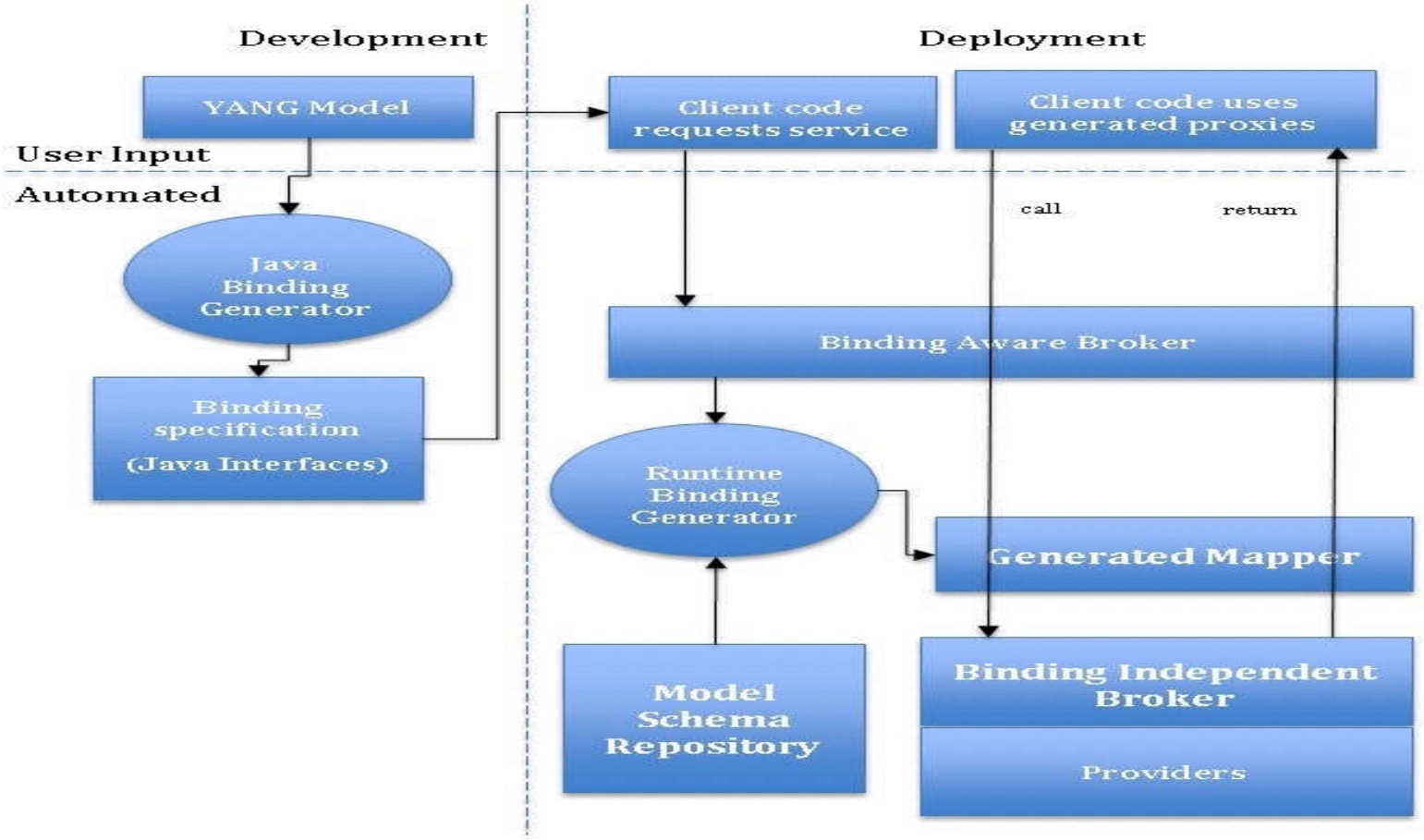




MD-SAL – Messaging Patterns

- **Unicast communication**
  - **RPC** - unicast between consumer and provider, where consumer sends **request** message to provider, which asynchronously responds with **reply** message
- **Publish / Subscribe**
  - Notifications - multicast message which is send by provider and is delivered to subscribed consumers
  - **Data Change Events** - multicast asynchronous event, which is sent by data broker if there is change in conceptual data tree, and is delivered to subscribed consumers
- **Data access patterns**
  - Transactional **reads** from conceptual **data tree** - read-only transactions
  - Transactional **modification** to conceptual **data tree** - write transactions
  - Transactional read and **modification** to conceptual **data tree** – read-write transactions
  - **Transaction Chaining**

MD-SAL – Data Access Patterns



DataTreeChangeListeners
Event Triggered only on the leader of the data store


```
DataTreeIdentifier <FlowCapableNode> id = new
DataTreeIdentifier(LogicalDatastoreType.OPERATIONAL, myInstanceId);

dataBroker.registerDataTreeChangeListener( id, myDataChangeListener) ;
```

ClusteredDataTreeChangeListeners Event Triggered on all the nodes


```
DataTreeIdentifier <FlowCapableNode> id = new
DataTreeIdentifier(LogicalDatastoreType.OPERATIONAL, myInstanceId);

dataBroker.registerDataTreeChangeListener( id, myDataChangeListener) ;
```

MD-SAL - Data Access Patterns (Contd.)

MD-SAL provides three different APIs to access data in the common data store:

Java generated DTOs – Binding Aware

```
NodeId nodeId = new NodeId("node1");
InstanceIdentifier instanceIdentifier = InstanceIdentifier.builder(Nodes.class)
        .child(Node.class, new NodeKey(nodeId)).toInstance();
```

DOM APIs – Binding Independent

```
QName nodes = QName.create("urn:opendaylight:inventory","2013-08-19","nodes");
QName node = QName.create(nodes,"nodes");
QName idName = QName.create(nodes,"id");
InstanceIdentifier instanceIdentifier =
org.opendaylight.yangtools.yang.data.api.InstanceIdentifier.builder().node(nodes).
nodeWithKey(node,idName,"node1").toInstance();
```

HTTP Restconf APIs

RESTCONF APIs can be used to access data:

http://<controller-ip>:8181/restconf/operational/opendaylight-inventory:nodes/

http://<controller-ip>:8181/restconf/config/opendaylight-inventory:nodes/opendaylight-inventory:node/node1/

Datastore – Transactions – Reading and Writing


```
ReadWriteTransaction transaction = dataBroker.newReadWriteTransaction();
Optional<Node> nodeOptional;

nodeOptional = transaction.read( LogicalDataStore.OPERATIONAL,
  n1InstanceIdentifier);

transaction.put(  LogicalDataStore.CONFIG,
  n2InstanceIdentifier,
  topologyNodeBuilder.build());

transaction.delete(  LogicalDataStore.CONFIG,
  n3InstanceIdentifier);

CheckedFuture future;
future = transaction.submit();
```

Datastore – Transactions – Merging


```
WriteOnlyTransaction transaction = dataBroker.newWriteOnlyTransaction();

InstanceIdentifier<Node> path =      InstanceIdentifier.create(NetworkTopology.class)
      .child(Topology.class, new TopologyKey("overlay1"));

transaction.merge( LogicalDataStore.CONFIG,  path,  topologyBuilder.build());

CheckedFuture future;
future = transaction.submit();
```

Exercise 2

Refer to the instructions in the ReadMe.txt under the directory /home/odl-developer/Tutorial in the VM

Integration with other features

- Services from other features (such as OpenFlowPlugin / Genius) can be utilized by
  - Invoking RPC
  - Invoking methods exposed through the API

- To bring in the dependency with other feature
  - Edit the impl/pom.xml with the API dependency of the other feature
  - In features/pom.xml – include the correct artifact-Id of the feature
  - In features/feature.xml –include the feature repository information. Include the API and IMPL dependency.
  - Inject the RPC / API in the blueprint XML.

GENIUS

- Genius project provides Generic Network Interfaces, Utilities & Services. Any ODL application can use these to achieve interference-free co-existence with other applications using Genius.
- In the first phase (ODL-Boron time frame), Genius would provide following modules --
- Modules providing a common view of Network interfaces for different services
  - **Interface (logical port) Manager**
    - *Allows bindings/registration of multiple services to logical ports/interfaces*
    - *Ability to plugin different types of southbound protocol renderers*
  - **Overlay Tunnel Manager**
    - *Creates and maintains overlay tunnels between configured TEPs*
- Modules providing commonly used functions as shared services to avoid duplication of code and waste of resources.
  - **Aliveness Monitor**
    - *Provides tunnel/nexthop aliveness monitoring services*
  - **ID Manager**
    - *Generates persistent unique integer IDs*
  - **MD-SAL Utils**
    - *Provides common generic APIs for interaction with MD-SAL*

Entity Ownership Service & Clustering Singleton

- The existing OpenDaylight service deployment model assumes symmetric clusters, where all services are activated on all nodes in the cluster.

- Many services require that there is a single active service instance per cluster these are 'singleton services'.

- The Entity Ownership Service (EOS) represents the base Leadership choice for one Entity instance. Every Cluster Singleton service *type* must have its own Entity and every Cluster Singleton service *instance* must have its own Entity Candidate.

- Every registered Entity Candidate should be notified about its actual role. All this "work" is done by MD-SAL so the OpenFowplugin need "only" to register as service in *SingletonClusteringServiceProvider* given by MD-SAL.

Exercise 3

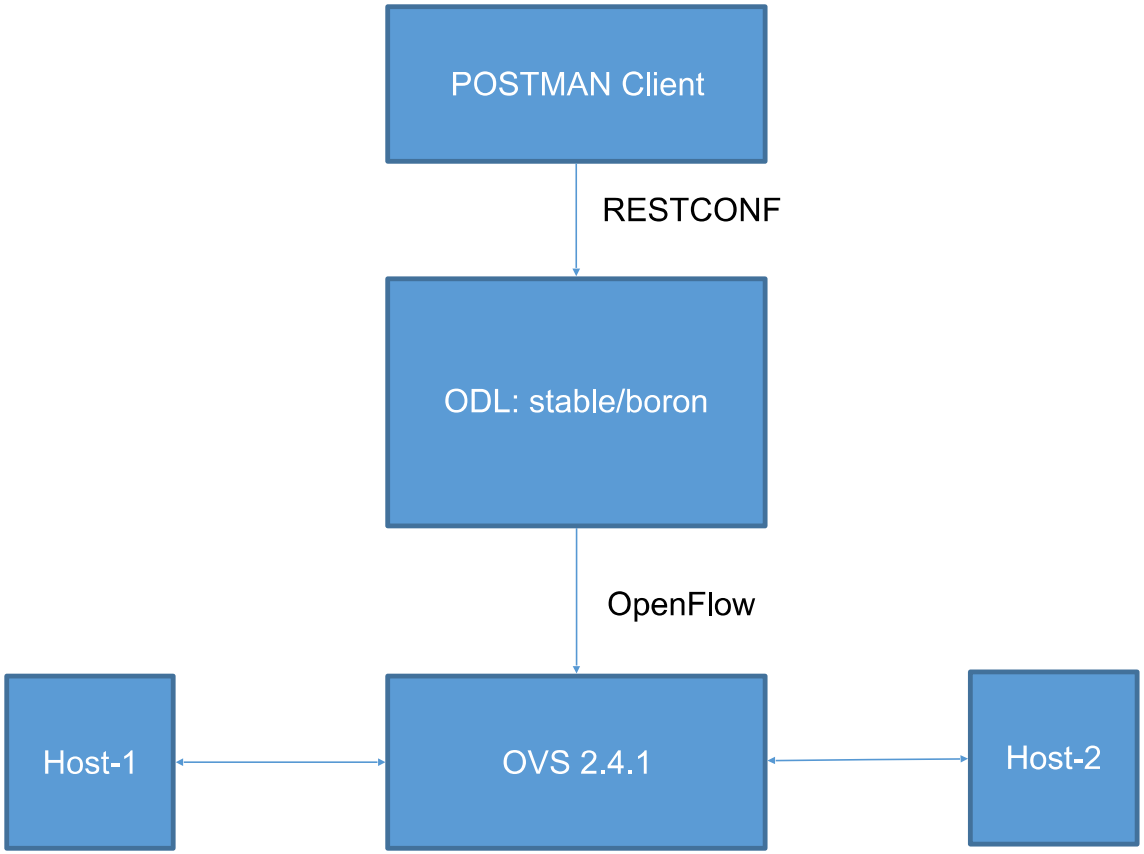
Refer to the instructions in the ReadMe.txt under the directory /home/odl-developer/Tutorial in the VM

Sample – Application Development

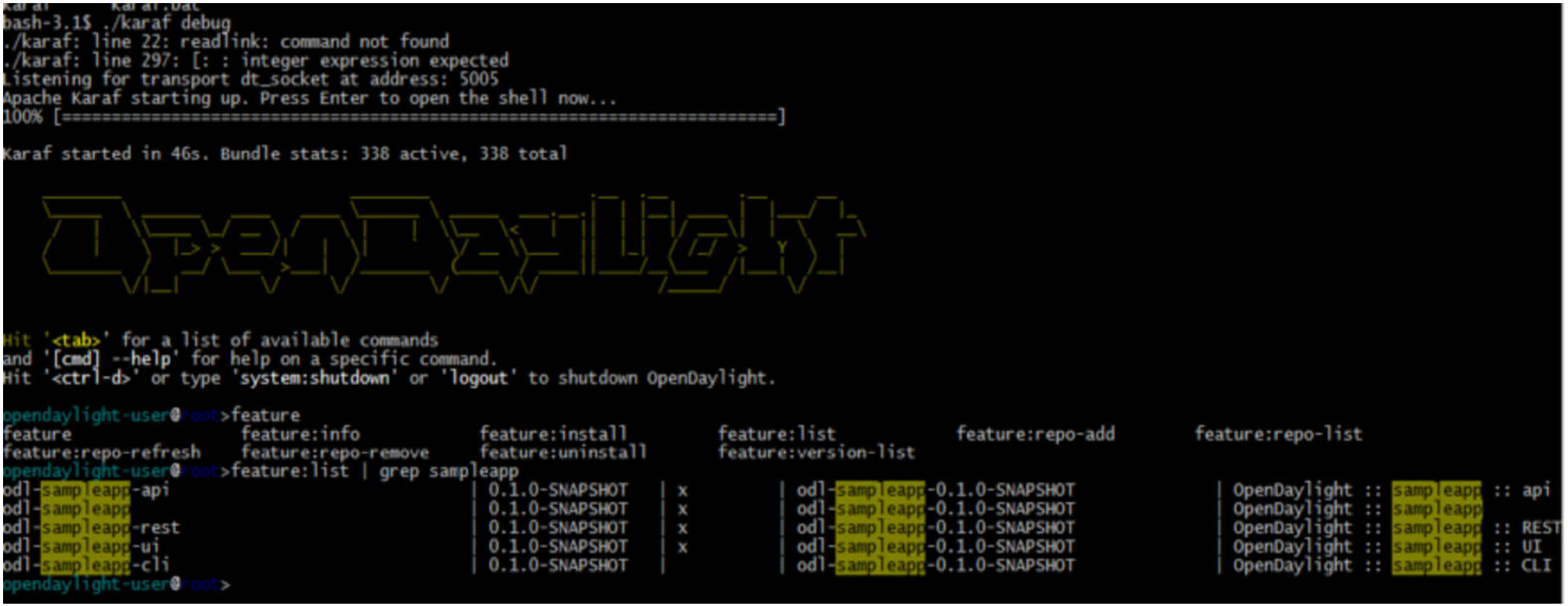
Test Topology

- The Topology Components
  - ODL Controller based on stable/boron with Sample App.
  - Mininet with OVS 2.4.1: OpenFlow Switch
  - POSTMAN: RESTCONF Client

Debugging


POSTMAN Client
RESTCONF
ODL: stable/boron
OpenFlow
Host-1
OVS 2.4.1
Host-2

Karaf – Installed Features



Communication between ODL Controller and OVS

- To initiate the OVS connection with ODL Controller, the below command is used:
  - sudo mn --controller=remote,ip=<Controller IP>

```
mininet@mininet-vm:~$ sudo mn --controller=remote,ip=100.96.2.33
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

```
[ebhgaur.IN00204647] > netstat -an | grep 6633
  TCP    0.0.0.0:6633           0.0.0.0:0              LISTENING
  TCP    100.96.2.33:6633       100.96.2.6:51516       ESTABLISHED
  TCP    [::]:6633              [::]:0                 LISTENING
```

Use Case: Deny Rule

- In this Use Case, the OVS will be programmed to perform a match on DST-IP and OF Action as DROP.
  - Sample RESTCONF Request to configure DENY Use Case:

```
{
  "access-list-entry": [
    {
      "rule-name": "deny",
      "node-id": "openflow:1",
      "matches": {
        "source-ipv4-network": "10.0.0.1/32",
        "destination-ipv4-network": "10.0.0.2/32"
       },
        "actions": {
        "deny": [
          null
        ]
      }
    }
  ]
}
```

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1 -O OpenFlow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x112000a, duration=2.834s, table=0, n_packets=0, n_bytes=0, priority=4,ip,nw_dst=10.0.0.2 actions=drop
```

Use Case: Permit Rule

- In this Use Case, the OVS will be programmed to perform a match on DST-IP and OF Action as Port Number where the Packet has to be forwarded.
  - Sample RESTCONF Request to configure PERMIT Use Case:

```
{
  "access-list-entry": [
    {
      "rule-name": "permit",
      "node-id": "openflow:1",
      "matches": {
        "source-ipv4-network": "10.0.0.1/32",
        "destination-ipv4-network": "10.0.0.2/32"
      },
        "actions": {
        "permit": [
          null
        ]
      }
    }
  ]
}
```

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1 -O OpenFlow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x112000a, duration=1.515s, table=0, n_packets=0, n_bytes=0, priority=5,ip,nw_dst=10.0.0.2 actions=write_actions(output:2)
 cookie=0x1230014, duration=1.511s, table=0, n_packets=0, n_bytes=0, priority=5,ip,nw_dst=10.0.0.1 actions=write_actions(output:1)
```

Data Plane Demo

- **Deny Use Case**
  - Static ARP Entries are added at Hosts H1 and H2
  - Flow added from SampleApp with Dst IP as Match and Drop as Action
  - Packets are hitting the Flow, since the action is drop, Ping Fail
- **Permit Use Case**
  - Static ARP Entries are added at Hosts H1 and H2
  - Flow added from SampleApp with Dst IP as Match and Permit as Action
  - Flow Entries for To and Fro traffic is provisioned
  - Packets are hitting the Flow, Since the flow entries have output port to push the packet, Ping is successful

Permit Use Case


```
root@mininet-vm:~# arp -s 10.0.0.2 8a:2a:2d:a5:bd:e9
root@mininet-vm:~#
```

Static ARP Entry at H1

```
root@mininet-vm:~# arp -s 10.0.0.1 fa:e0:c6:7c:23:fa
root@mininet-vm:~#
```

Static ARP Entry at H2

Flow Entry For Permit Action

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.150 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.117 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.047 ms
```

Ping H2 from H1

```
root@mininet-vm:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
06:52:19.400486 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1938, seq 99, leng
th 64
06:52:19.400520 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1938, seq 99, length
 64
06:52:19.857668 LLDP, length 71: openflow:1
06:52:20.400326 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1938, seq 100, len
gth 64
06:52:20.400379 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1938, seq 100, lengt
```

TcpDump at H1

```
root@mininet-vm:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
06:52:09.400209 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1938, seq 89, leng
th 64
06:52:09.400222 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1938, seq 89, length
 64
06:52:09.856177 LLDP, length 71: openflow:1
06:52:10.401277 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1938, seq 90, leng
th 64
06:52:10.401305 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1938, seq 90, length
 64
```

TcpDump at H2

Deny Use Case

```
root@mininet-vm:~# arp -s 10.0.0.2 8a:2a:2d:a5:bd:e9
root@mininet-vm:~#
```

Static ARP Entry at H1

```
root@mininet-vm:~# arp -s 10.0.0.1 fa:e0:c6:7c:23:fa
root@mininet-vm:~#
```

Static ARP Entry at H1

Flow Entry for Deny Action

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
```

Ping H2 from H1

```
root@mininet-vm:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
06:56:44.880238 LLDP, length 71: openflow:1
06:56:45.348281 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2028, seq 12, leng
th 64
06:56:46.348201 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2028, seq 13, leng
th 64
06:56:47.349115 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2028, seq 14, leng
th 64
06:56:48.348239 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2028, seq 15, leng
th 64
06:56:49.348226 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2028, seq 16, leng
th 64
```

TcpDump at H1

```
root@mininet-vm:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
06:56:49.880540 LLDP, length 71: openflow:1
06:56:54.880954 LLDP, length 71: openflow:1
```

TcpDump at H2

OpenFlow Plugin – using FRM/RPC

Using FRM

- The ACL Rule is committed to the Application's Config Datastore through Northbound (Eg. RestConf)
- The Application Listens to the Data Change for that Tree and pushes Flow Rule to Inventory Config DS.
- Forwarding Rules Manager listens for the Flow Data Tree change Notification
- FRM configures the Flow Rule to device through OpenFlowPlugin and OpenFlowJava (Serialize/ De-Serialize)

Advantage :- The flow rule is persisted in the Inventory Config datastore. FRM reconciliation pushes the flow to the device which upon restart,

OpenFlow Plugin – using FRM/RPC

Using RPC

- The Rule is committed to the Application;s Config Datastore through Northbound (Eg. Restconf)
- The Application Listens to the Data Change for that Tree and uses the RPC exposed by OpenFlowPlugin Models
- The RPC pushes the Flow Rule to device through OpenFlowPlugin and OpenFlowJava (Serialize/ De-Serialize) – Bypassing the FRM

- Advantage :- Provides better performance as RPC is directly used to push the flow rule.
- Disadvantage :- Flow rules are not persisted in the controller and so upon restart the application needs to take care of pushing the flow again.

Best Practices

- Datastore transactions are expensive and optimal usage of datastore is very important.
- Avoid unnecessary Config / Oper DS reads and writes as they incur high transactional cost
- Applications should use Future Callbacks rather than the blocking Future.get().
- Applications must subscribe to CDTCN at appropriate level in the conceptual data-tree to performance impact when processing CDTCL during normal conditions / switch restart / cluster restart scenarios

Best Practices (Contd.)

- Applications must not do heavy processing in the context of the CDTCL thread and execute in another thread context forked from CDTCL.

- Applications can cache the data when processing CDTCL fired the config DS. This will give better performance as the costly remote DS reads can be avoided.

- Applications that run on cluster and needs better performance should use CDTCL rather than DTCL as latter is fired only in the leader-node of a shard and leader can change during normal operations of the cluster

Thank You