Master of Science in Internetworking

Capstone Project Report

# Deployment of SDN Controller and configuring Juniper devices using NETCONF

Borzoo Khosravi

Supervisor

Mr. Shahnawaz Mir

Winter 2023

# Contents

# Introduction

In different periods of networking history, there were always main challenges that engineers had to find a solution for. In the early days of networking, the main challenges were to define standards and protocols that help networks to get widespread acceptance and contribution by companies and manufacturers. Years after that, challenges move toward designing and deploying a network that is more reliable with higher performance and bandwidth capacity. In those days, servers and networks were still static. Once established, a network topology was not expected to change much, if at all. Applications were associated with a single server that had a fixed location on the network. These applications used the network to exchange information with other applications that were similarly in fixed locations [1]. This situation with the advent of server virtualization technologies and microservice architecture has changed. Today, network applications are dynamic in nature [1]. They can move between servers. Number of network changes required to meet today's network application requirements have been drastically increased. It is impossible to cover this level of changes with the traditional approach of a network admin manually configuring all the devices in a network one by one. As another example, old network architectures usually were limited to on-prem[1] infrastructure, consisted of limited number of zones such as DMZ[2] zone or Internal zone. Each zone contains servers that independently provide a specific service for users such as ERP[3], VOIP, Accounting applications and etc. But, applications in today's network have hybrid architectures. In addition, using microservice architecture to develop applications results in, service components that create an application to be distributed over multiple places such as cloud and on-prem servers while, based on old day approach everything resides on a single location on a single server. It is evident that, the dynamicity and complexity level of networks have increased. In order to meet both the current/future requirement of network infrastructure and ease network operation/maintenance, today's networks need to be agile, dynamic and flexible.

Automating is the only way to achieve this goal. There are different ways to automate network operation/configuration. In this project we will take a look at SDN solution, specifically OpenDaylight as an open-source SDN and how to use NETCONF protocol to manage Juniper devices via OpenDaylight SDN.

# 1. What is SDN

Software Defined Networking (SDN) is a new paradigm in network architecture that promises to bring about significant benefits in terms of flexibility, scalability, and cost efficiency. SDN main objective is the separation of control plane and forwarding plane. [2][3] In SDN terminology the

[1] On-premises or on-prem denote when network device and servers are installed and run on resources available on the premises of a person or organization, rather than at a remote facility such as cloud

[2] demilitarized zone

[3] Enterprise resource planning

party responsible for control plane is called “Controller entity” and the party responsible for Forwarding plane is called “Controlled entity”. [2] Communication between controller entity and controlled entity happens through **interfaces**. But what is an **interface** in SDN terminology?

“A point of interaction between two entities. When the entities are placed at different locations, the interface is usually implemented through a network protocol. If the entities are collocated in the same physical location, the interface can be implemented using a software application programming interface (API), inter-process communication (IPC), or a network protocol.” [2]

## 1.1 SDN Architecture and components

Whenever talked about SDN, it is referred to a framework or an architecture. SDN may include different layers and span across multiple planes. (Figure 1.1) Each plane or layer has a specific functionality that will be discussed later and can communicate internally or with each other through interfaces. Interfaces in the below diagram are shown with "Y".

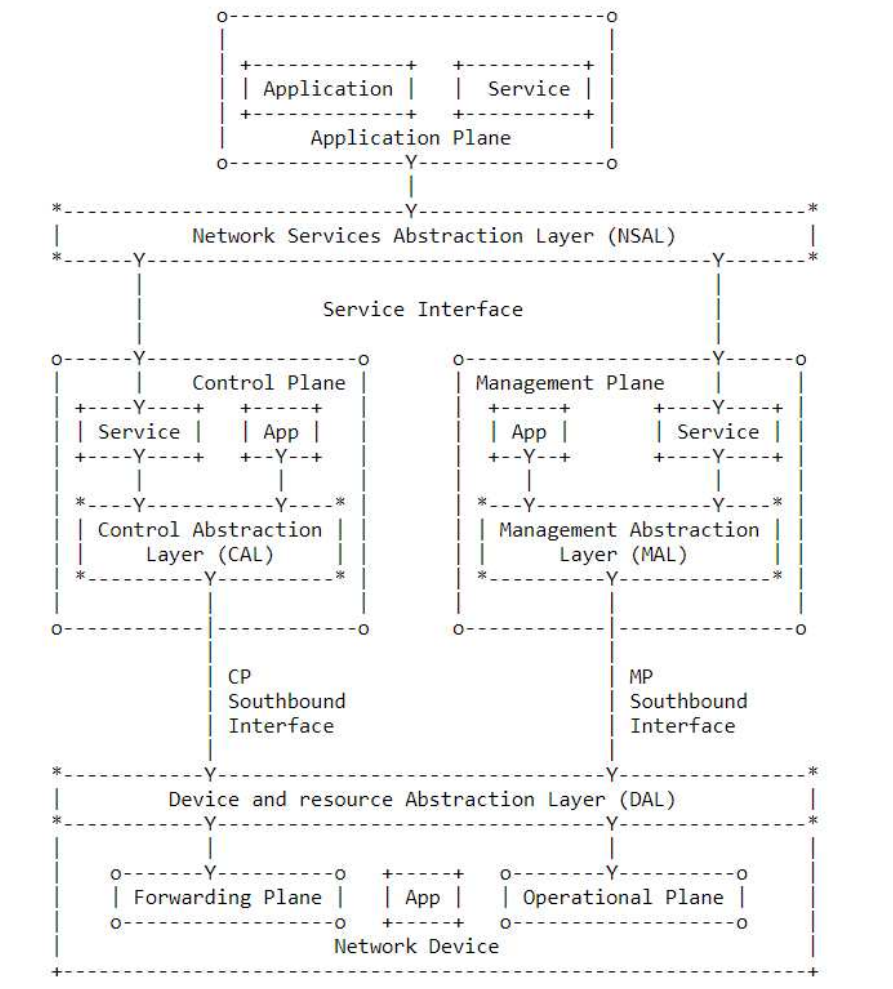



*Figure 1-1.1 SDN Layer Architecture Source: https://www.rfc-editor.org/rfc/rfc7426*

The main components of SDN architecture include [2]

- **Forwarding Plane** - Responsible for forwarding, dropping and changing of packets that passes through a controlled entity based on the instruction that is received from controller entity. The forwarding plane is usually the termination point for control-plane services and applications. The forwarding plane is also referred to as the "data plane" or the "data path".

- **Operational Plane** – Based on RFC7426 it is the termination point for management plane services and applications. It is responsible for managing operational state of the network device. By operational state, it means anything related to port status, memory status, traffic info and so on. In a wide range of network devices there is no independent operational plane. In most cases, the operational plane and Forwarding plane are merged together.

- **Control Plane** – This plane's main responsibility is to make decision how packets should be forwarded by one or more controlled entity (network device). This is achieved through fine-tuning Forwarding-table in forwarding plane based on the network topology or external service request. Although control plane is supposed to deal with Forwarding plane, in some cases, it may use the operational plane data, such as port state or port load, to make better packet forwarding decisions.

- **Management Plane** – Based on RFC7426, the main responsibilities of this plane are to monitor, configure and maintain network devices. That means all the decisions about the state of the network device and its ports are supposed to happen in this plane.

- **Application Plane** – All the application and services that can control network device behavior reside in this plane. Monitoring applications, configuration application and so on.

Layered architecture of SDN controllers and use of standard protocols for communication between these layers makes it possible for different contributors and vendors to focus on a specific part of their interest. That can be from creating different types of applications to monitor or configure network device behavior, to develop interfaces or core components of SDN controllers, or even develop YANG modules used for communication between SDN controllers and network device. YANG will be discussed later on a separate section.

Now that we know what SDN is, the architecture of SDN and its main components, the next question is what are the advantages and disadvantages of using SDN controllers?

## 1.2 Advantages of SDN controller

1. Flexibility: SDN allows network administrators to program the network with a centralized view of the entire network, which enables them to manage the network more efficiently. SDN also allows administrators to quickly adjust the network to changing business requirements, which is particularly important for cloud-based applications and services. [4] [5]

2. Scalability: SDN makes it easier to scale the network, since it abstracts the control plane from the data plane, allowing administrators to manage a larger number of devices with less effort. This is particularly important in cloud environments where resources are rapidly added or removed. [4]

3. Security: SDN makes it easier to implement security policies, since it allows administrators to define and enforce policies at a central location, rather than implementing them on each individual device.

4. Cost Efficiency: SDN can reduce costs by allowing administrators to automate many network management tasks, such as provisioning and configuration, which can reduce the need for expensive human resources. [4]

5. Improved Network Visibility: SDN provides a centralized view of the network, which allows administrators to monitor network traffic and identify potential issues before they become problems. This is particularly important for organizations that rely on real-time data. [5]

## 1.3 Disadvantages of Software Defined Networking (SDN)

1. Complexity: SDN introduces new layers of abstraction and complexity, which can be difficult to manage, particularly for organizations that do not have the expertise to deploy and manage SDN. [4]

2. Single Point of Failure: SDN introduces a central controller that is responsible for managing the network, which creates a single point of failure. If the controller fails, the network could be at risk of downtime or other issues.

3. Limited Vendor Support: While SDN has gained traction in the industry, many vendors have been slow to adopt the technology. This can limit the number of options available to organizations that want to deploy SDN.

4. Security Risks: SDN can introduce new security risks, particularly if the central controller is not properly secured. This can create a vulnerability that could be exploited by attackers.

5. Skills Gap: SDN requires specialized skills that may not be readily available in the industry. This can make it difficult for organizations to find and hire qualified personnel to manage and maintain SDN.

## 1.4 OpenDaylight

OpenDaylight or ODL is an open-source project managed by Linux foundation. Project started back in 2013 with over 40 members including big companies such as Brocade, Cisco, Citrix, Ericsson, IBM, Juniper Network, Microsoft, NEC, Red Hat and VMware. [6]

ODL is based on the Java programming language and follows a loosely coupled modular structure. Loosely coupled modules are a collection of independent modules that each have a specific purpose and usage and are able to communicate with each other through different protocols like APIs, RPCs and so on. According to OSGI (the Open Services Gateway Initiative), group of these modules can gather and create a large application like OpenDaylight [7]

"Similar to other controllers we have seen, ODL also supports network programmability via southbound protocols, a bunch of programmable network services, a collection of northbound APIs, and a set of applications." [7]

### 1.4.1 ODL architecture

ODL architecture may vary on different versions but the core concept and architecture between different versions is identical. Figure 1.2 shows a simplified ODL architecture.

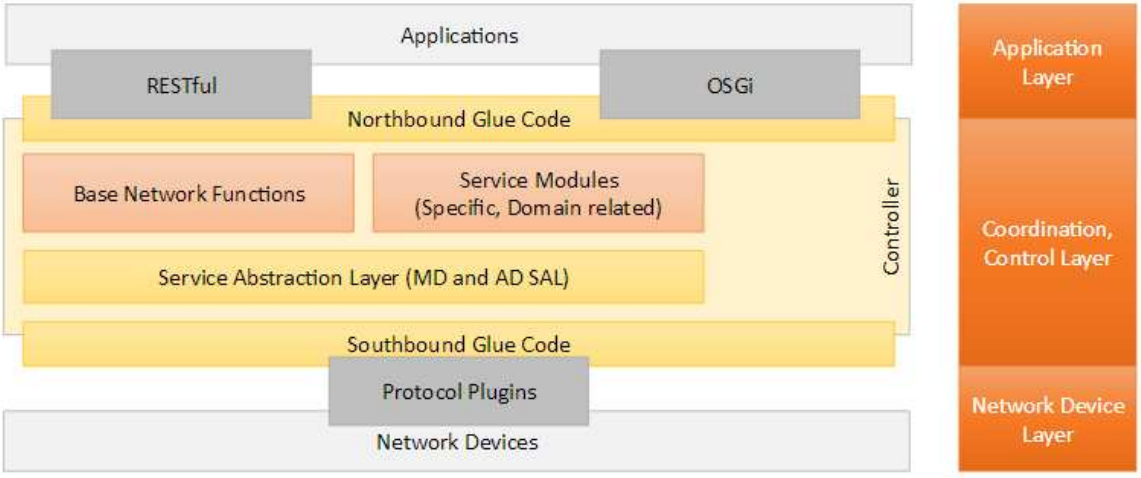



*Figure 1-2 Simplified ODL architecture source: https://thenewstack.io/sdn-series-part-vi-opendaylight/*

ODL architecture mainly consists of 3 layers. [7]

1. Network device layer (Southbound plugins and protocols).
2. coordination and control layer (Service adaptation and network functions).
3. application layer (Northbound APIs and applications).

#### 1.4.1.1 Southbound plugins and protocols

Southbound plugins support multiple protocols to communicate with network devices, including NETCONF, OpenFlow, BGP-LS, LISP, SNMP, etc. Each of these protocol plugins will independently link to Service Abstraction Layer (SAL) either MD or AD SAL. Then it is the duty of SAL to fulfill all the applications' requests regardless of what the southbound protocol is.

#### 1.4.1.2 Service Abstraction Layer and Network functions

**BNSF:**

Base Network Service Functions or BNSFs in the control layer of ODL are responsible for collecting information and gathering statistics about the elements within the network and their capabilities. BNSFs will expose these data to consumer applications through Northbound APIs [8][9].

Some base network service functions that come shipped with ODL are:

1. **Topology manager:** provide network node and node connectors (switch ports) detail
2. **Statistic manager:** collect statistical information from managed nodes
3. **Switch manager:** provide network node and node connectors (switch ports) detail
4. **Forwarding rule manager:** manage forwarding rules such as OpenFlow rules

**SAL:**

This layer is the heart of ODL. With help of SAL, ODL can support multiple southbound protocols (via Southbound plugins) and can provide a uniform set of services to other modules and Northbound applications. SAL provide a service called Device Discovery that can be consumed by the Topology Manager to form the network topology and to build element capabilities. Most of SAL services are built based on the SB plugins features. the requested service for a given switch is fulfilled by the SAL, irrespective of the underlying SB protocol [9]. Northbound and Southbound plugins in ODL can be either service producers or service consumers or even both. Here the role of SAL is to act as a large services registry point where the producers advertise their service via their APIs. Then after a consumer requests an advertised service by a generic API, the SAL connects and binds both the service producer and consumer [7].

There are two types of SAL in ODL. AD-SAL and MD-SAL.

The developers of ODL started coding the original SAL with an API-Driven SAL architecture. The problems of AD-SAL architecture are:

1. Developer had to code SAL APIs to route service request between consumers and providers
2. Developer had to code adaptation functionality whenever Northbound (service, abstract) API is not similar to corresponding Southbound (protocol) API [9].

In MD-SAL architecture to route the data between consumers and producers, java uniform APIs which are generated from YANG models are used. [10]

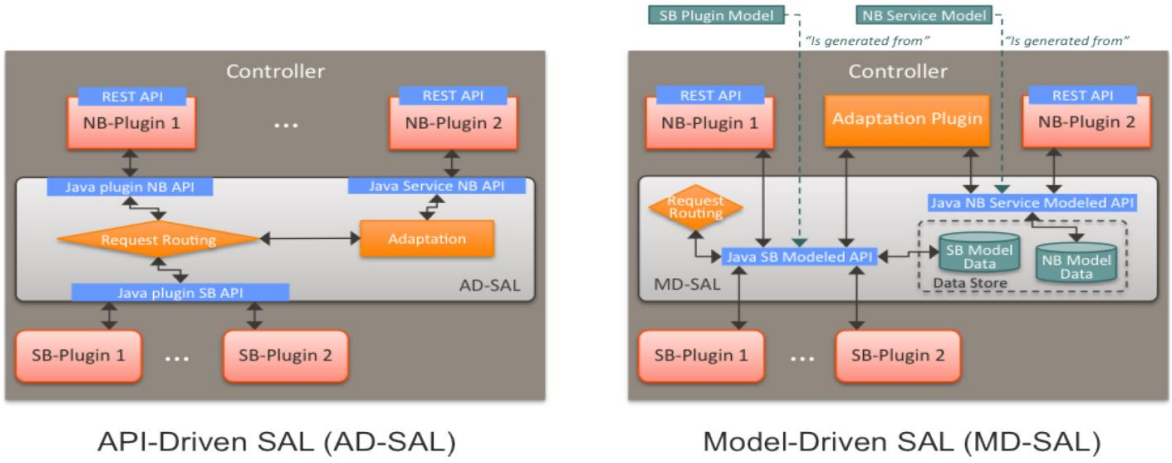



*Figure 1-3 Evolution of SAL from API-Drivne to Model-Driven*
*source:https://www.researchgate.net/publication/317057083_Facilitation_of_The_OpenDaylight_Architecture*

### 1.4.1.3 application layer (Northbound APIs and applications)

All applications that are consumer of any type of services provided by different modules of ODL reside in this layer. For application to consume services ODL exposes northbound APIs via two different architecture.

1. Open Services Gateway Initiative (OSGi) architecture which is mainly used by the applications that run on the same address space as controller. [7]
2. REST APIs which can be used by application that run on the same machine as the controller or on a different machine. One such application is Postman that in the 4.3.2 (Configuring NETCONF devices with RESTCONF call from OpenDaylight), it is shown how we can use it to communicate with ODL. [7]

# 2. YANG

In this chapter and the next one, YANG modeling language and NETCONF protocol will be explained.
NETCONF/YANG is the IETF solution for network automation. YANG provides a language to describe your desired configuration (or state). NETCONF, on the other hand provides the protocol to deliver and perform the required operations in order to achieve the desired state, described within the YANG model. Figure 2.0.1 better explains how NETCONF/YANG is used in network automation. [11]

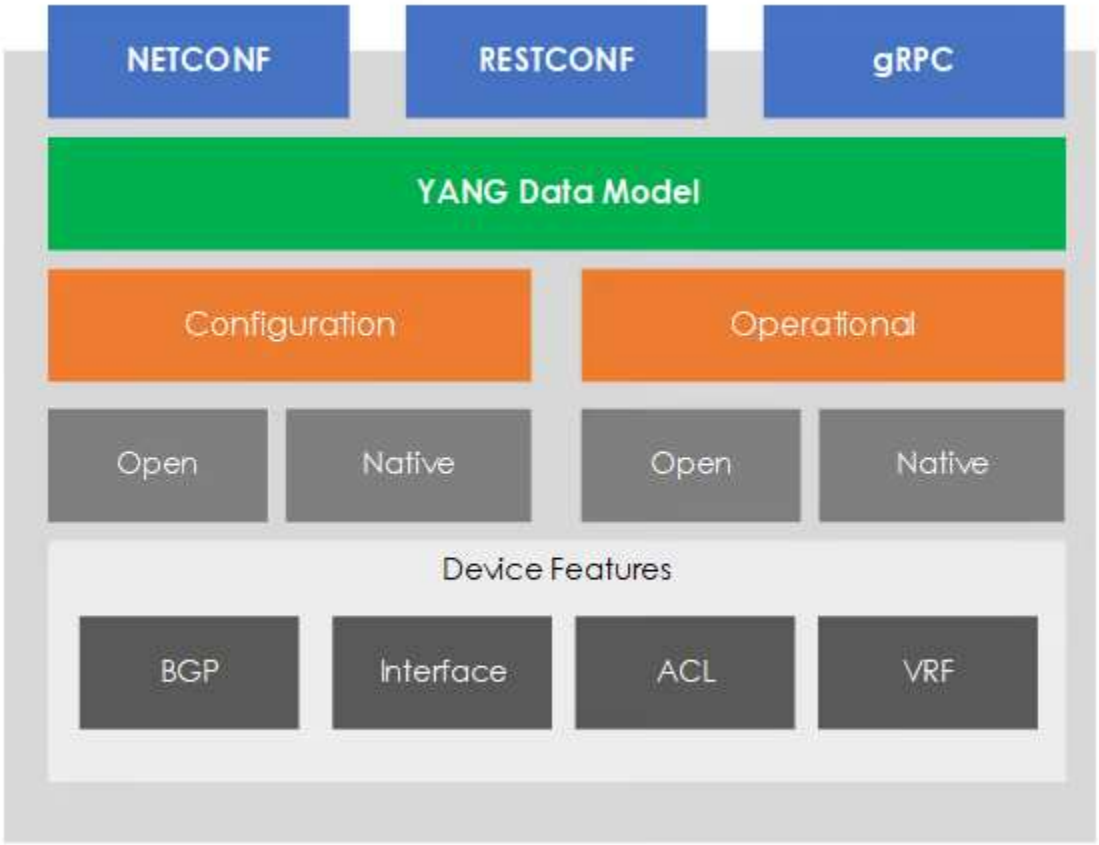



*Figure 2.0.1 NETCONF/YANG Stack - source: https://www.fir3net.com/Networking/Protocols/an-introduction-to-netconf-yang.html*

## 2.1 What is YANG

"YANG is a data modeling language for the definition of data sent over network management protocols such as the NETCONF and RESTCONF." [12]
But what does it mean? Basically, the data that is about to be passed between two entities over a standard protocol needs to have a structure. YANG is a tree structure format that can pass data over different protocols such as NETCONF or RESTCONF.

YANG is a standard tree structure that can model different types of data, including: [13]

1-Configuration data
2- State data
3- Event notifications that are sent by network elements
4-Model NETCONF RPCs (Remote Procedure Call)

## 2.2 What are the advantages of using YANG in the NETCONF protocol?

YANG is a data modeling language that is published and well-documented by Internet Engineering Task Force (IETF) in RFC6020 and later in RFC7950. It means that when different vendors create YANG modules that define how to retrieve operational data or configure their network devices, they use the same standard data modeling format. This makes it possible for different types of automation tools and SDN solutions to be able to read these YANG schemas from various types of network devices and know how to communicate with them.

Another advantage of the YANG modeling language is that it is protocol independent. YANG data structures can convert to other types such as JSON or XML. Later in chapter 4 (Juniper Automation) it is shown that the OpenDaylight RESTCONF module will take the NETCONF device's operational and configuration data in YANG structure but with the format of XML or JSON.

"YANG is structured into modules, where in general, modules consist of three main elements; module-header statements, revision statements, and definition statements. Header statements, as one would expect, define the name of the module, the modules namespace on the network device, the module definition, or any modules imported from a different YANG model. The revision statement simply gives the history about the module. And finally, the most robust, the definition statement is actually the main body that defines the data model." [14]

## 2.3 YANG node types

To define a data model based on RFC6020, YANG uses four types of node. [15]

1. Leaf Nodes
2. Leaf-List Nodes
3. Container Nodes
4. List Nodes

### 2.3.1 Leaf and Leaf-list nodes

In RFC6020 leaf is a data node that exists in at most one instance in the data tree. A leaf has a value but no child nodes. Leaf-list is like the leaf node but defines a set of uniquely identifiable nodes rather than a single node. Again, in here each node also has a value but no child nodes. Let's take a look at the differences by a sample.

YANG snippet code to define a flag "enable" with a default value of True: [16]

```
leaf enabled {
     type boolean;
     default true;
}
```

Example of XML instance:

```
<enabled>false</enabled>
```

YANG code to model a list of ciphers [16]

```
leaf-list cipher {
     type string;
}
```

Example of XML instance:

```
<cipher>blowfsh-cbc</cipher>
<cipher>3des-cbc</cipher>
```

### 2.3.2 Containers and Lists

There are two other types of nodes in YANG, "container" and "list". A list node can have numerous instances, whereas a container node can only have one. Different key values distinguish different instances in a list.
Container definition based on RFC6020:
"container: An interior data node that exists in at most one instance in the data tree. A container has no value, but rather a set of child nodes." [15]

Following wo examples show the difference between container and list node.

Container sample code: [16]

```
container timeout {
     leaf access-timeout {
          type uint32;
     }
```

```
    leaf retries {
        type uint8;
        }
}
```

Example instance of the above code:

```
<timeout>
    <access-timeout>60</access-timeout>
    <retries>2</retries>
</timeout>
```

List sample code: [16]

```
list user {
    key "login-name";
    leaf login-name {
        type string;
    }
    leaf full-name {
        type string;
    }
}
```

Example of XML instance:

```
<user>
    <login-name>hakanm</login-name>
    <full-name>Hakan Millroth</fullname>
</user>
<user>
    <login-name>mbj</login-name>
    <full-name>Martin Bjorklund</fullname>
</user>
```

Figure 2.3.2.1 shows a sample YANG file opened by pyang tools. Also, different types of node are shown in this picture.

Other terminologies are used in YANG modeling language, well-documented in RFC6020 and RFC7950. For further study refer to the following RFCs.

https://datatracker.ietf.org/doc/html/rfc6020#section-3
https://datatracker.ietf.org/doc/html/rfc7950#section-3

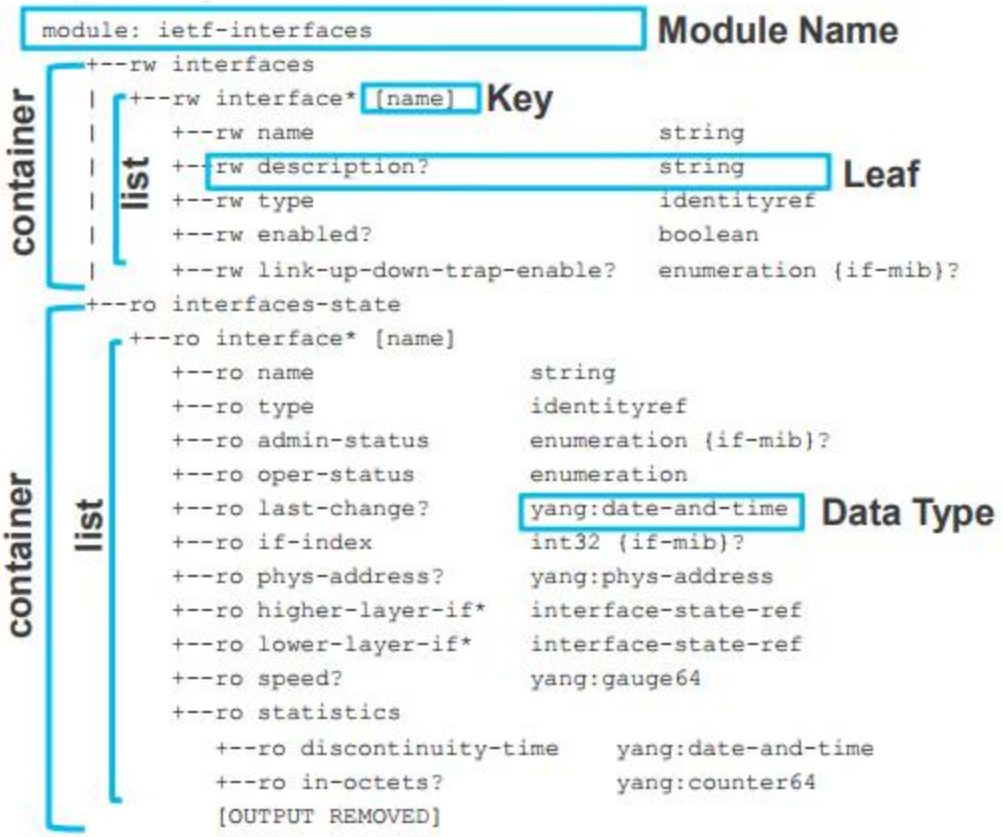



*Figure 2.3.2.1 Sample YANG file opened by pyang tools source: http://YANG.ciscolive.com/pod0/labs/lab2/lab2-m1*

## 2.4 Junos YANG modules

Repository of all YANG modules used for different versions of Juniper operating system (Junos) is located at the following URL:

https://github.com/Juniper/YANG

Junos have different directories for each version of Junos. Inside each of these directories there are sub-directories based on the Junos family. To check which Junos family match our NETCONF device, simply use the following command:

***root@Juniper-router> show system information***
***Model: vmx***
***Family: Junos*** <---------- this is the family version
***Junos: 20.1R1.11*** <---------- this is the firmware version
***Hostname: Juniper-router***

***root@Juniper-router>***

Junos YANG modules are categorized based on the type of NETCONF requests. All the YANG modules that are related to configuration data start with:

***Junos-conf-****

And they are all located under **/Junos/conf/** directory. (Figure 2.4.1)

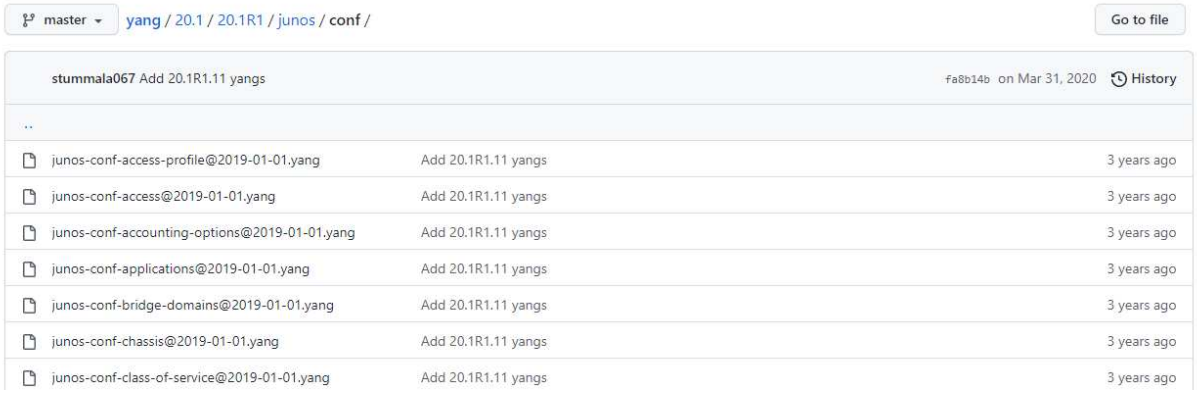


*Figure 2.4.1 Junos configuration YANG modules*

All the YANG modules that are related to state data (Operational Data) are located under the **/Junos/rpc/** directory and their name will start with:

***Junos-rpc-****

Later at section 4.3 (Juniper automation using OpenDaylight and NETCONF) we will take a look at one of these YANG modules in the MG-SOFT YANG explorer application.

# 3. NETCONF

## 3.1 History of NETCONF

The NETCONF base protocol was officially published as a RFC 4741 NETCONF Configuration Protocol in late 2006. Depending on the transport layer protocol there are different supporting RFCs for each of them. [17]

- RFC 4742 Using the NETCONF Configuration Protocol over Secure Shell (SSH) [20]
- RFC 4743 Using NETCONF over the Simple Object Access Protocol [21]
- RFC 5539 NETCONF over Transport Layer Security (TLS) [22]

In 2011, two new RFC were introduced which obsoleted RFC 4271 and 4272 respectively. [17]

- RFC 6241 obsoletes RFC 4741 [19]
- RFC 6242 obsoletes RFC 4742 [23]

## 3.2 Why NETCONF is needed in today's networks? Disadvantages of CLI and SNMP

Automating network configuration requires both framework and standard protocol which all vendors can use to facilitate network automation on their device. SDN will act as the framework and NETCONF is the protocol used for communication to network devices.

Back to old days, CLI scripting was the primary approach to making automated configuration changes to the network prior to NETCONF. CLI scripting has several limitations including:
**1- Lack of transaction management** [18]
That means if you want to change multiple items on a network device with a single script, there is no mechanism to guarantee whether all items were successfully applied to device or if there was failure which one failed. For example, imagine we want to add a route plus change an interface IP address and also edit an ACL with a single script. Normally when one part of the script fails, then all the rest will fail too. That is due to a lack of transaction management in cli scripting approach.

**2- No structured error management**
It means if you try to use a script to configure a device and something unexpected happen you will not get a standard format and meaningful response back like when you are working with an API or NETCONF RPC message.

**3- CLI Syntax Changes [17]**
Syntax of commands will change and may vary even across different products of the same vendor. As an example, there are differences in the syntax of firmware on cisco routers and cisco switches. that makes scripts fragile and costly to maintain.

These are all consequences of the fundamental reality that CLIs are created for human usage rather than an API for programmed access.

Prior to NETCONF, the SNMP protocol was used. This protocol was intended to read data from devices or even write modifications, but in reality, it is primarily used for performance and monitoring applications. There are number of reasons that SNMP was never accepted as a network configuration protocol:

1- The lack of a clearly defined discovery procedure, which makes it challenging to locate the appropriate MIB modules.

2- UDP protocol's structural constraints, no reliable and ordered data transmission. [18]

3- The absence of practical standard security and commit methods. SNMP lack authentication and authorization mechanism to ensure messages are transmitted securely.

4- SNMP manages device configuration on a per-device basis and does not support network-level configuration or multi-device configuration collaboration. [17]

## 3.3 NETCONF protocol Framework

Based on RFC5241, NETCONF protocol has a hierarchical structure consist of 4 layers (Figure3.3.1). Each layer will encapsulate data from higher layer and provide service to its upper layer.

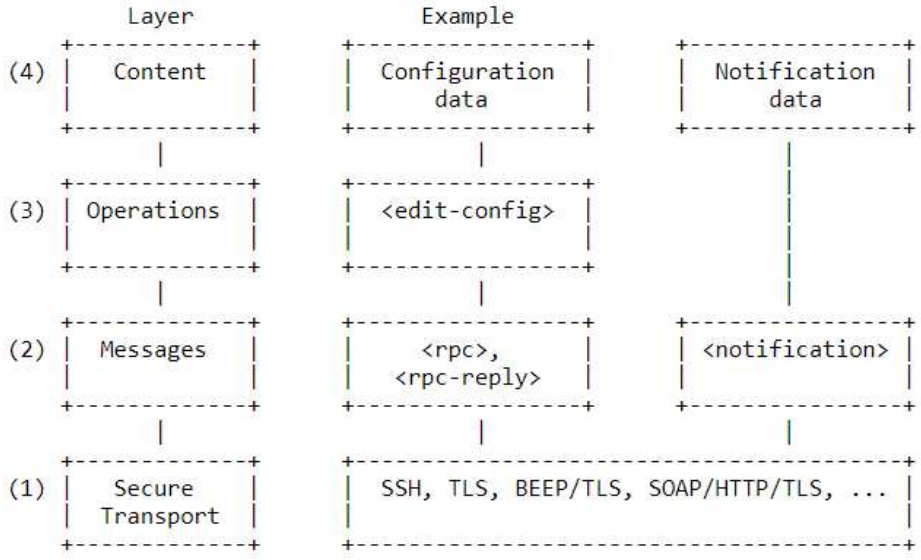



*Figure 3.3.1 NETCONF protocol Framework – Source: http://YANG.ciscolive.com/pod0/labs/lab2/lab2-m1*

**NETCONF layers functionalities:** [18] [19]

- **Secure Transport layer**
  This layer provides a communication path between the client and server. NETCONF can be layered over any transport protocol that meets basic requirements.
  Secure Shell (SSH) is the preferred transport protocol in NETCONF for transmitting XML information. Currently, Huawei devices support SSH as the transport protocol of NETCONF.
- **Messages layer**
  This layer provides a simple, transport-independent framing mechanism for encoding RPCs and notifications.
  A client encapsulates an RPC request into an <rpc> element and sends it to a server. The server encapsulates the result of processing this request into an <rpc-reply> element and sends it to the client.
- **Operations layer**
  This layer defines a set of base protocol operations, which are invoked as RPC methods with XML-encoded parameters.
- **Content layer**
  This layer is defined by a data model that manages data. Currently, mainstream data models include Schema and YANG.

The following picture shows the structure of a complete NETCONF YANG request and how each part of the message maps to different layers of NETCONF framework.

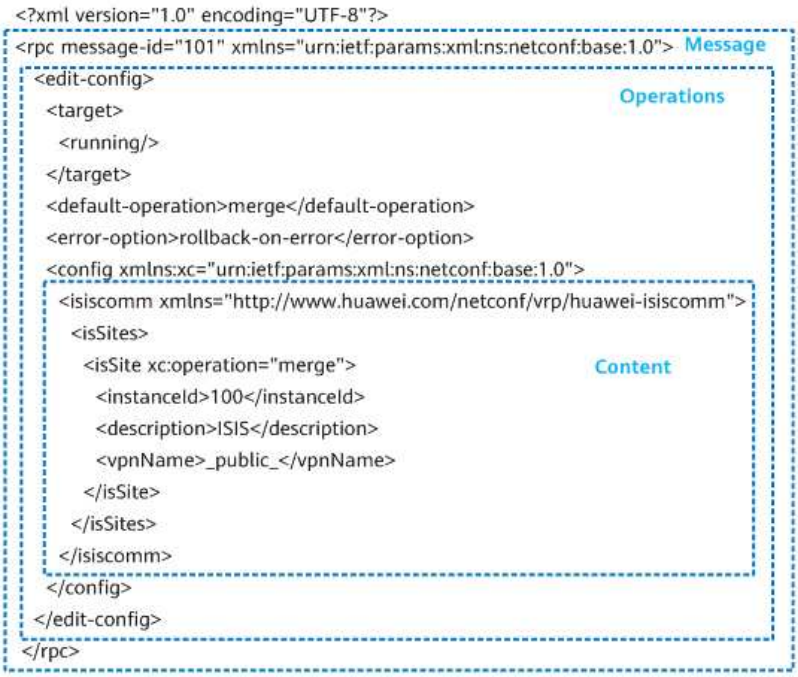



*Figure 3-1Sample NETCONF YANG request and how it maps to NETCONF framework*
*Source:http://YANG.ciscolive.com/pod0/labs/lab2/lab2-m1*

## 3.4 How does NETCONF protocol work?

NETCONF operates in a client-server model, where the client is a network management system (NMS) and the server is the network device. The NMS sends a request to the network device using NETCONF, and the network device responds with the requested configuration data. NETCONF uses XML as its data encoding format, which enables the configuration data to be structured and hierarchical. This makes it easier to parse and manipulate the data programmatically.

Client and server in NETCONF protocol provides the following functions: [18]

- NETCONF client:
  - Manages network devices using NETCONF.
  - Sends RPC requests to a NETCONF server to query or modify one or more parameter values.
  - Learns the status of a managed device based on the alarms and events sent by the
- NETCONF server:
  - Maintains information about managed devices and responds to client-initiated requests.
  - When receiving a request from a NETCONF client, the NETCONF server parses the request and sends a reply to the client.
  - If a fault or another type of event occurs on a managed device, the NETCONF server reports an alarm or event to the client through the notification mechanism. This allows the client to learn the status of the managed device.

### NETCONF session consist of 3 main parts:

1- **Session Establishment:** NETCONF client and server establish an SSH session over a predefined TCP port (default 830), after passing the authentication and authorization phase, each side will send the hello messages to inform the other party of its capabilities.

2- **Operation Request:** After NETCONF client learns about server capabilities, it can send a series of rpc requests (operations) to the NETCONF server via <rpc> messages and server will respond with <rpc-reply> messages.

3- **Session Close:** The client then ask to close the connection by <close-session> message.

### NETCONF operations:

Depending on the NETCONF server's capabilities, a client can perform different kinds of actions on the NETCONF server. Following is a list of operations included in base protocol, other operations availability will depend on the advertised capabilities that NETCONF server sends via hello messages at the beginning of a NETCONF session.

*Table 3.4.1 NETCONF base protocol operations - source: https://www.rfc-editor.org/rfc/rfc6241*

| Operation | Description |
|---|---|
| **<get>** | Retrieve running configuration and device state information. |
| **<get-config>** | Retrieve all or part of a specified configuration datastore. |
| **<edit-config>** | Load all or part of a configuration to the specified configuration data store |
| **<copy-config>** | Replace the whole configuration data store with another data store |
| **<delete-config>** | Delete a configuration data store |
| **<lock>** | allows the client to lock the entire configuration datastore system of a device |
| **<unlock>** | The <unlock> operation is used to release a configuration lock, previously obtained with the <lock> operation. |
| **<close-session>** | Request graceful termination of a NETCONF session. |
| **<kill-session>** | Force the termination of a NETCONF session. |

# 4. Juniper automation

In this part, different approaches in Junos automation will be explained.
The Junos operating system has advantages in terms of automation compared to the operating systems of other companies that manufacture network equipment. Other vendors like Cisco have experience changes in their firmware over time and now have multiple firmware that may not be consolidated across different products such as Cisco routers, switches and firewalls. But in case of Junos, all Juniper devices regardless of their type use the same operating system (Junos) with features that may or may not be available based on the type of device. That will simplify automation process because, the developer/automation admin need to deal with only a single YANG model[1] to configure multiple devices or very similar YANG models[2]. Later it is shown how all requests that are sent to the network devices use a RESTCONF call with the body that can differ based on the task.

Junos is inherently built with automation in mind. Even when a command is sent to it through the CLI, this operating system first converts it to XML RPC format and then sends it to the internal kernel. [24]

"The Junos OS command-line interface (CLI) and the Junos OS infrastructure communicate using XML. When you issue an operational mode command in the CLI, the CLI converts the command into XML format for processing. After processing, Junos OS returns the output in the form of an XML document, which the CLI converts back into a readable format for display. Remote client applications also use XML-based data encoding for operational and configuration requests on devices running Junos OS." [24]

All Junos configuration statements and most of Junos operational statements that can be issued from CLI by the admin, have equivalent XML API requests and if required XML responses.

In the following sections, we will look at different ways we can automate Juniper device configuration.
Junos supports both REST API and NETCONF for automation. First let's briefly examine how to configure Junos operating system with REST API and later Junos automation with NETCONF will be discussed.

## 4.1 Junos configuration with REST API

Before using REST API for configuration, it is required to enable REST API on Juniper Device.

```
set system services rest http port 8080
set system services rest http addresses 192.168.1.2
set system services rest enable-explorer
```

[1] Assuming all the device that are required to be configured for a specific task are using a single Junos family and version.
[2] Assuming all the device that are required to be configured for a specific task are using different Junos family and versions.

These three commands will enable REST API with basic configuration (HTTP port and IP address to respond. Junos REST API Explorer is also enabled with the third command. Later, we will see how it looks and work.

After submitting above three commands then we should be able to connect to Junos rest explorer based on the port and IP address we have set on the device.

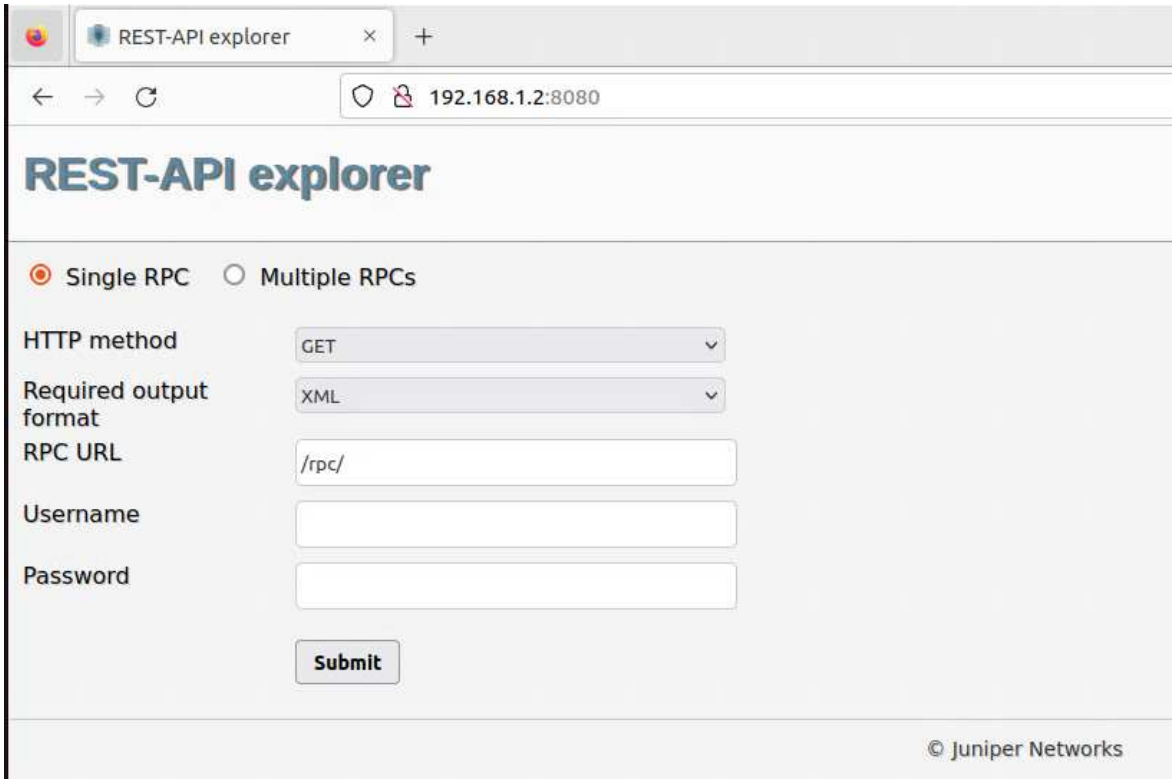



*Figure 4.1.1 Screenshot of REST-API explorer web page on lab Junos Device*

Using Junos REST-API explorer we can send REST API requests and see the output in the response body. Following picture is an example of REST API call (**GET /rpc/get-system-information**) with Juniper REST Explorer.

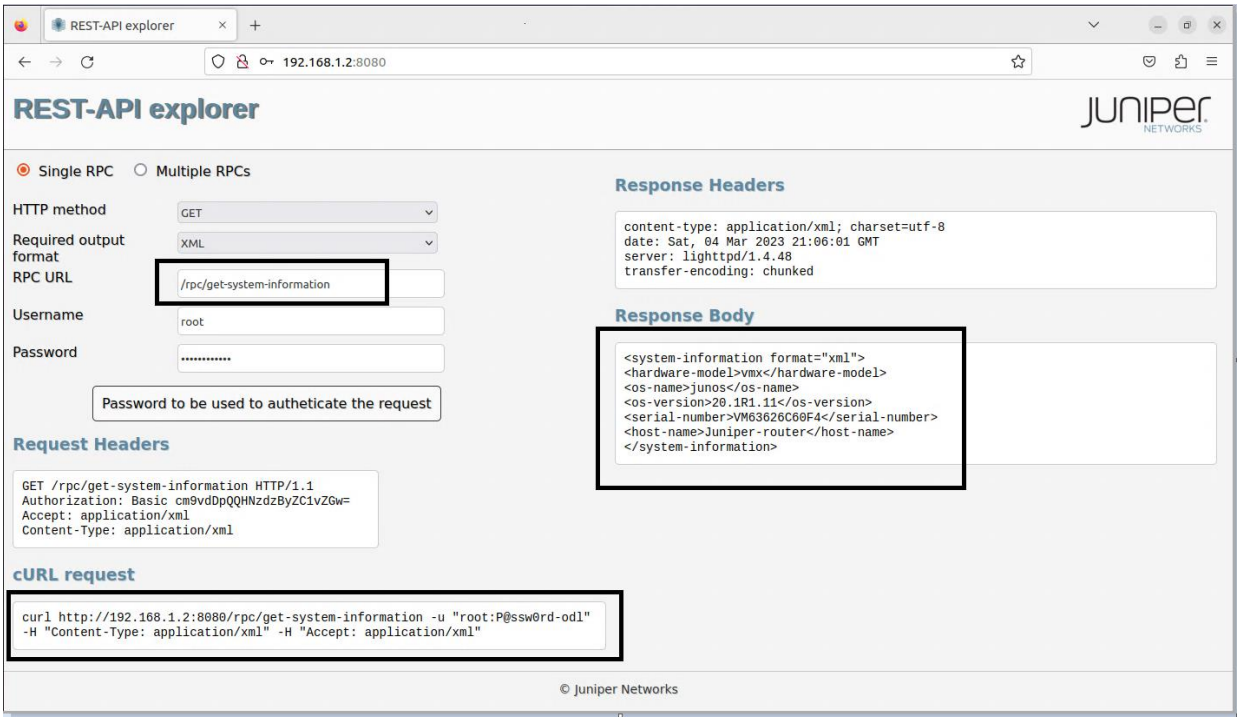



*Figure 4.1.2 Screenshot of "get-system-information" rpc call with REST-API explorer*

Above picture shows how we can get system information from device.
One of the cool features of REST-API explorer is that it will also show the equivalent curl request, which we can embed in any automation tool to achieve the same result. Figure 4.1.3 shows the curl request copied from REST-API explorer in the previous step and run directly from the shell console of OpenDaylight server.

```
odl-admin@odl-server:~$ curl http://192.168.1.2:8080/rpc/get-system-information -u "root:P@ssw0rd-odl" -H "Content-Type: application/xml" -H "Accept: applicat
ion/xml"
<system-information format="xml">
<hardware-model>vmx</hardware-model>
<os-name>junos</os-name>
<os-version>20.1R1.11</os-version>
<serial-number>VM63626C60F4</serial-number>
<host-name>Juniper-router</host-name>
</system-information>
odl-admin@odl-server:~$
```

*Figure 4.1.3 Requesting the same information as figure 4.1.2 but this time with curl.*

After copying the cURL request from REST-API explorer and running it on the network management server, the result will be exactly the same as what is shown in the REST-API explorer.

Earlier, it was mentioned that Juniper Junos convert all CLI commands to XML RPC calls in the background and get the response back from Junos core in the format of XML RPC.
Then it will convert it back to CLI syntax and represent it to CLI user. Any command entered on Junos CLI has an equivalent XML RPC call. The easiest way to find the equivalent XML RPC call on Junos cli is to use **pipe** after each command followed by "**display xml rpc**". For example we will use the following command to get Junos system information from CLI.

root@Juniper-router> show system information

```
root@Juniper-router> show system information
Model: vmx
Family: junos
Junos: 20.1R1.11
Hostname: Juniper-router

root@Juniper-router>
```

*Figure 4.1.4 "show system information" cli command output*

If we use **pipe** and then **display xml rpc**, we can find the equivalent REST API RPC call

```
root@Juniper-router> show system information | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/20.1R0/junos">
    <rpc>
        <get-system-information>
        </get-system-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>

root@Juniper-router>
```

*Figure 4.1.5 "show system information | display xml rpc" output*

It indicates that to get the equivalent rpc request of any Junos cli operation command, we need to pipe the command with “display xml rpc” added to the end of it. Output will show the desired rpc request. When we use this rpc request to fetch operational data from Junos device, result will be the same as the output of cli command (in this case “**show system information**”) [25]

For more information about JUNOS REST API, take a look at the following link:

https://www.juniper.net/documentation/us/en/software/Junos/rest-api/index.html

## 4.2 Junos configuration with NETCONF

In this part we will look at how we can set up NETCONF over SSH on a Juniper Device and then will try to get system configuration from device. After that we will take a look at how we can use NETCONF to change device hostname, change interface IP address and add/remove static routes with NETCONF.

### 4.2.1 Enabling NETCONF service over SSH on Junos device (NETCONF server)

RFC 4742, Using the NETCONF Configuration Protocol over Secure Shell (SSH), requires that the NETCONF server, by default, provide the client device with access to the NETCONF SSH subsystem when the SSH session is established over a dedicated IANA-assigned TCP port. Use of a dedicated port makes it easy to identify and filter NETCONF traffic. The IANA-assigned port for NETCONF-over-SSH sessions is 830. [26]

In order to enable NETCONF on Juniper device and use it later with OpenDaylight, execute the following command on the Juniper device.

***set system services ssh root-login allow***

This will allow root account to connect to Juniper device with SSH protocol. This approach is used because of simplicity. But the best practice is to create a local, radius or TACACS account on Juniper device and assign required permission to that user. Then use that user instead of root account to connect remotely via SSH NETCONF.

Next line will enable NETCONF access over SSH protocol on port 830, which is the default NETCONF port.

***set system services netconf ssh port 830***

Configure the NETCONF server to enforce certain behaviors that are compliant with RFC 4741, NETCONF Configuration Protocol, during NETCONF sessions. [27]

***set system services netconf rfc-compliant***

Configure the **yang-compliant** statement to require that the NETCONF server return YANG-compatible configuration data for the <get-config> and <get-configuration format="xml"> RPCs. [27]

***set system services netconf yang-compliant***

Following 5 commands will enable traceoption for NETCONF transaction that are made to Juniper device. Using "**show log netconf-ops.log**" we can see the detail of all NETCONF transactions. [27]

***set system services netconf traceoptions file netconf-ops.log***
***set system services netconf traceoptions file size 3m***
***set system services netconf traceoptions file files 20***
***set system services netconf traceoptions file world-readable***
***set system services netconf traceoptions flag all***

The next step is to connect to Juniper device with NETCONF over SSH from the control manager server.

In this scenario, control manager server is the ODL (OpenDaylight) server. This can be any server that has access to the port 830 of Juniper devices. With the following SSH command, we can connect to Juniper device via NETCONF SSH.

**ssh <*username*>@<*IP-address*> -p 830 -s netconf**

After entering password, Juniper device will send NETCONF hello element message which conforms to RFC6241 and will include a list of NETCONF supported capabilities.

```
odl-admin@odl-server:~$ ssh root@192.168.1.2 -p 830 -s netconf
(root@192.168.1.2) Password:
<!-- No zombies were killed during the creation of this user interface -->
<!-- user root, class super-user -->
<nc:hello xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:capabilities>
   <nc:capability>urn:ietf:params:netconf:base:1.0</nc:capability>
   <nc:capability>urn:ietf:params:netconf:capability:candidate:1.0</nc:capability>
   <nc:capability>urn:ietf:params:netconf:capability:confirmed-commit:1.0</nc:capability>
   <nc:capability>urn:ietf:params:netconf:capability:validate:1.0</nc:capability>
   <nc:capability>urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,file</nc:capability>
   <nc:capability>urn:ietf:params:xml:ns:netconf:base:1.0?module=ietf-netconf&amp;revision=2011-06-
01</nc:capability>
   <nc:capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0</nc:capability>
   <nc:capability>urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0</nc:capability>
   <nc:capability>urn:ietf:params:xml:ns:netconf:capability:validate:1.0</nc:capability>
   <nc:capability>urn:ietf:params:xml:ns:netconf:capability:url:1.0?scheme=http,ftp,file</nc:capability>
   <nc:capability>urn:ietf:params:xml:ns:yang:ietf-inet-types?module=ietf-inet-types&amp;revision=2013-
07-15</nc:capability>
   <nc:capability>urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring</nc:capability>
   <nc:capability>http://xml.juniper.net/netconf/Junos/1.0</nc:capability>
   <nc:capability>http://xml.juniper.net/dmi/system/1.0</nc:capability>
  </nc:capabilities>
  <nc:session-id>60122</nc:session-id>
</nc:hello>
]]>]]>
```

Note that each NETCONF request/response message will end with "]]>]]>"

Most of Juniper YANG models fall into two main categories:

1- YANG model for operational commands
2- YANG models for configuration

An example of operational command will be the "**show arp**" command and later we will see configuration command examples such as changing interface IP or device hostname.

First let's get back to cli and see the command format to get ARP table.

```
root@Juniper-router> show arp
MAC Address       Address         Name                Interface           Flags
00:0c:29:ae:55:2e 128.0.0.16      fpc0                em1.0               none
00:0c:29:d0:15:98 192.168.1.1     192.168.1.1         fxp0.0              none
00:0c:29:8f:b8:4a 192.168.1.3     192.168.1.3         fxp0.0              none
00:0c:29:f8:66:07 192.168.1.4     192.168.1.4         fxp0.0              none
00:0c:29:2e:2f:e7 192.168.1.5     192.168.1.5         fxp0.0              none
Total entries: 5
```

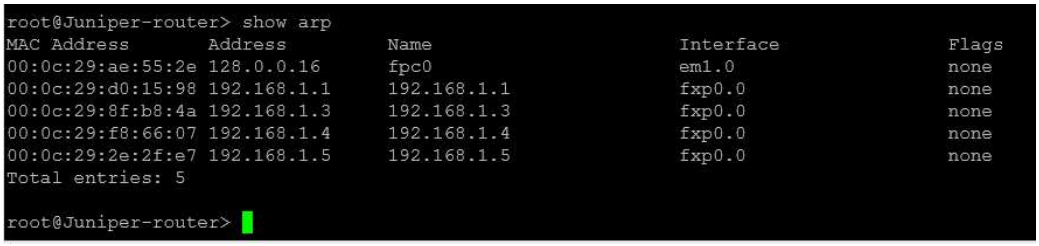



*Figure 4.2.1.1 "show arp" cli output*

To execute the command with NETCONF protocol, we need to know the equivalent YANG model and RPC call. One way to find the RPC call is by using the method mentioned earlier at the end of REST API section, by adding pipe followed by "**display xml rpc**" to then end of the operational command.

```
root@Juniper-router> show arp | display xml rpc
<rpc-reply xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
  <rpc>
    <get-arp-table-information>
    </get-arp-table-information>
  </rpc>
  <cli>
    <banner></banner>
  </cli>
</rpc-reply>

root@Juniper-router>
```

From the output, it is clear that in order to get ARP table result from a NETCONF RPC call, we need to execute "**get-arp-table-information**" RPC.
To send the operational command using NETCONF we will wrap the operational command inside rpc xml tag.

```
<rpc>
```

```
        <get-arp-table-information/>
</rpc>
```

And execute it under NETCONF SSH console.
**Note:** it is better to end NETCONF messages with "**]]>]]>**" but not mandatory in this case.

The output will be like this:

```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
<arp-table-information xmlns="http://xml.juniper.net/Junos/20.1R0/Junos-arp"
Junos:style="normal">
.
.
.
Full output at Appendix B (7.1)
.
.
.
<arp-table-entry>
<mac-address>
00:0c:29:d0:15:98
</mac-address>
<ip-address>
192.168.1.1
</ip-address>
<hostname>
192.168.1.1
</hostname>
<interface-name>
fxp0.0
</interface-name>
<arp-table-entry-flags>
<none/>
</arp-table-entry-flags>
</arp-table-entry>
<arp-table-entry>
<mac-address>
00:0c:29:8f:b8:4a
</mac-address>
<ip-address>
192.168.1.3
</ip-address>
<hostname>
192.168.1.3
</hostname>
<interface-name>
fxp0.0
```

</interface-name>
<arp-table-entry-flags>
<none/>
</arp-table-entry-flags>
</arp-table-entry>
.
.
Full output at Appendix B (7.1)
.
.
<arp-entry-count>
5
</arp-entry-count>
</arp-table-information>
</nc:rpc-reply>
]]>]]>

We can even narrow down the result to ARP entries for a specific interface. Look at the following example

```
root@Juniper-router> show arp interface em1.0 | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/20.1R0/junos">
    <rpc>
        <get-arp-table-information>
                <interface>em1.0</interface>
        </get-arp-table-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>

root@Juniper-router>
```

*Figure 4.2.1.2 "show arp interface em1.0 | display xml rpc" output*

root@Juniper-router> show arp interface em1.0 | display xml rpc
<rpc-reply xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
  <rpc>
    <get-arp-table-information>
        <interface>em1.0</interface>
    </get-arp-table-information>
  </rpc>
  <cli>
    <banner></banner>
  </cli>
</rpc-reply>

root@Juniper-router>

Copying the highlighted part over a NETCONF SSH session will show the following result

```
<rpc>
        <get-arp-table-information>
                <interface>em1.0</interface>
        </get-arp-table-information>
    </rpc>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="http://xml.juniper.net/junos/20.1R0/junos">
<arp-table-information xmlns="http://xml.juniper.net/junos/20.1R0/junos-arp" junos:style="normal">
<arp-table-entry>
<mac-address>
00:0c:29:ae:55:2e
</mac-address>
<ip-address>
128.0.0.16
</ip-address>
<hostname>
fpc0
</hostname>
<interface-name>
em1.0
</interface-name>
<arp-table-entry-flags>
<none/>
</arp-table-entry-flags>
</arp-table-entry>
</arp-table-information>
</nc:rpc-reply>
]]>]]>
```

*Figure 4.2.1.3 NETCONF rpc request to retrieve interface em1.0 ARP entries*

```
<rpc>
    <get-arp-table-information>
        <interface>em1.0</interface>
    </get-arp-table-information>
  </rpc>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
<arp-table-information xmlns="http://xml.juniper.net/Junos/20.1R0/Junos-arp"
Junos:style="normal">
<arp-table-entry>
<mac-address>
00:0c:29:ae:55:2e
</mac-address>
<ip-address>
128.0.0.16
</ip-address>
<hostname>
fpc0
</hostname>
<interface-name>
em1.0
</interface-name>
<arp-table-entry-flags>
<none/>
```

```
</arp-table-entry-flags>
</arp-table-entry>
</arp-table-information>
</nc:rpc-reply>
]]>]]>
```

They are displayed in different colors to facilitate distinguishing the request from the response. But, how can we find in which YANG model this RPC method is defined?
As mentioned earlier, the Junos YANG models mostly fall into operation or configuration categories. Since this task (getting arp table entries) is an operational task, the YANG model file should start with **Junos-rpc**[1] and if it was a configuration task YANG model should start with **Junos-conf** [2]. Ok now that we know this, how can we find which Junos-rpc YANG file is related to retrieving ARP table entries?

One way to find it, is directly from Junos cli. By issuing the following command

***show system schema module ?***

In the output we need to search for any schema that may include "arp" such as the one in the following picture:

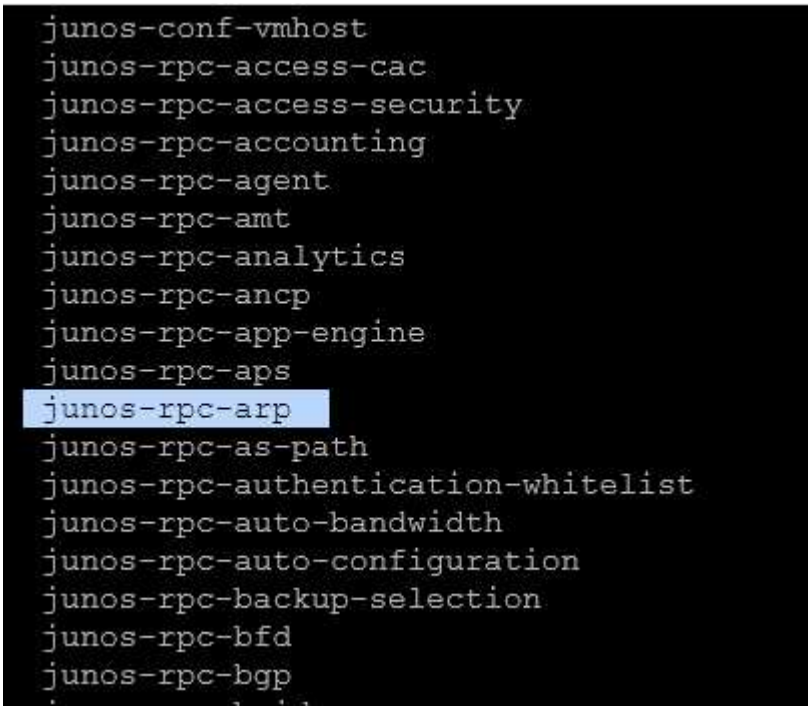



*Figure 4.2.1.4 "show system schema module ?" cli output*

[1] Refer to 2.4 Junos YANG modules section for more info

[2] Refer to 2.4 Junos YANG modules section for more info

In the next step, we need to download the YANG file starting with this name from Junos github repository from the following repository:

https://github.com/Juniper/yang

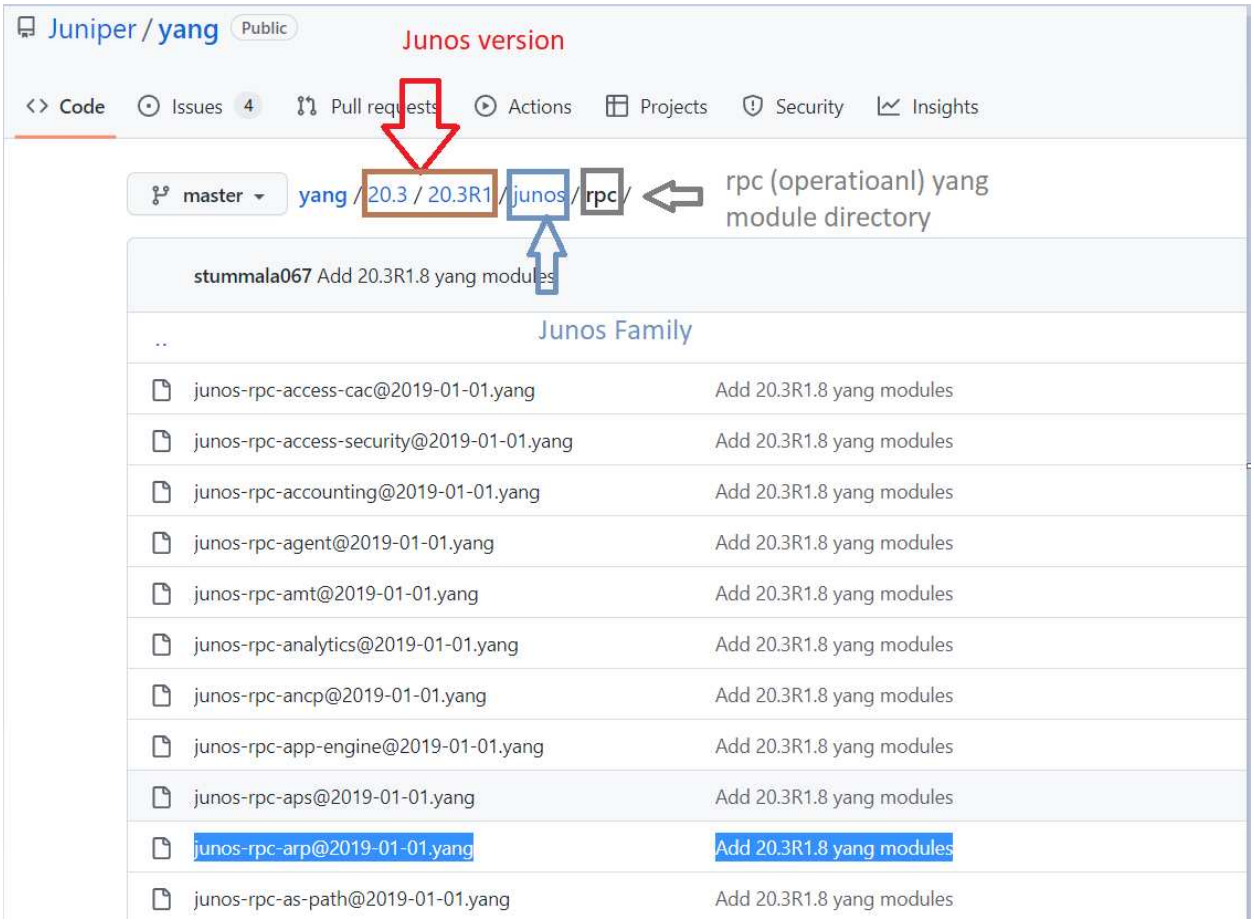



*Figure 4.2.1.5 junos-rpc-arp YANG module file on github repository*

To have a better understanding of the YANG file structure, after downloading we can use different YANG parsing tools to see the structure of this YANG file. In this case we use MG-SOFT YANG explorer to open this YANG file. (Figure 4.2.1.6)

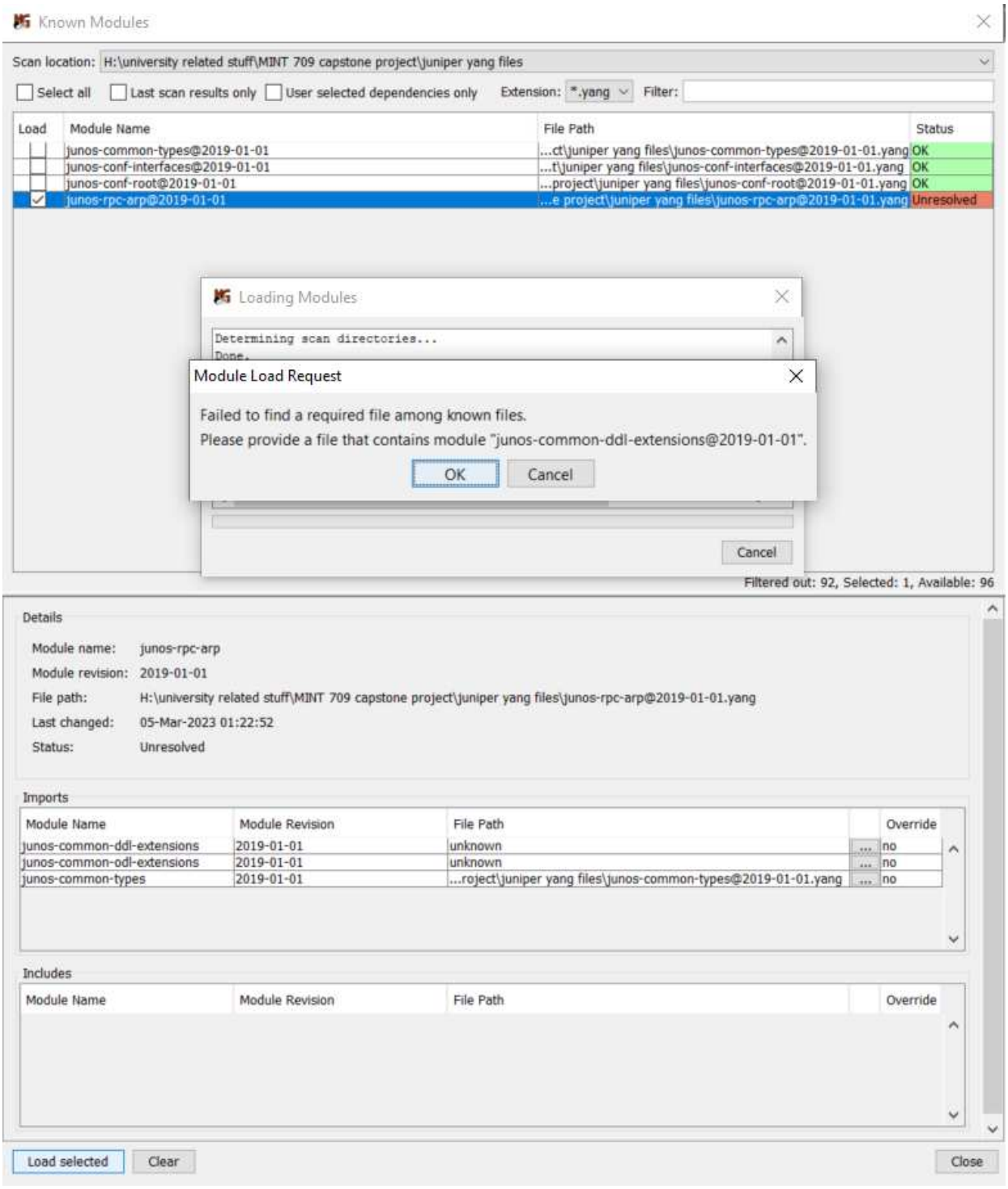



*Figure 4.2.1.6 Opening "junos-rpc-arp" yang file failed because of missing another YANG file.*

Trying to load the downloaded YANG file into MG-SOFT explorer generated an error message. Because it requires other YANG modules, which are imported into this YANG module and two of them are missing on the downloading system.

**Junos-common-ddl-extensions**
**Junos-common-odl-extensions**

After downloading these two modules and loading them into MG-SOFT YANG Explorer now we can see the "Junos-rpc-arp@2019-01-01" YANG file structure.

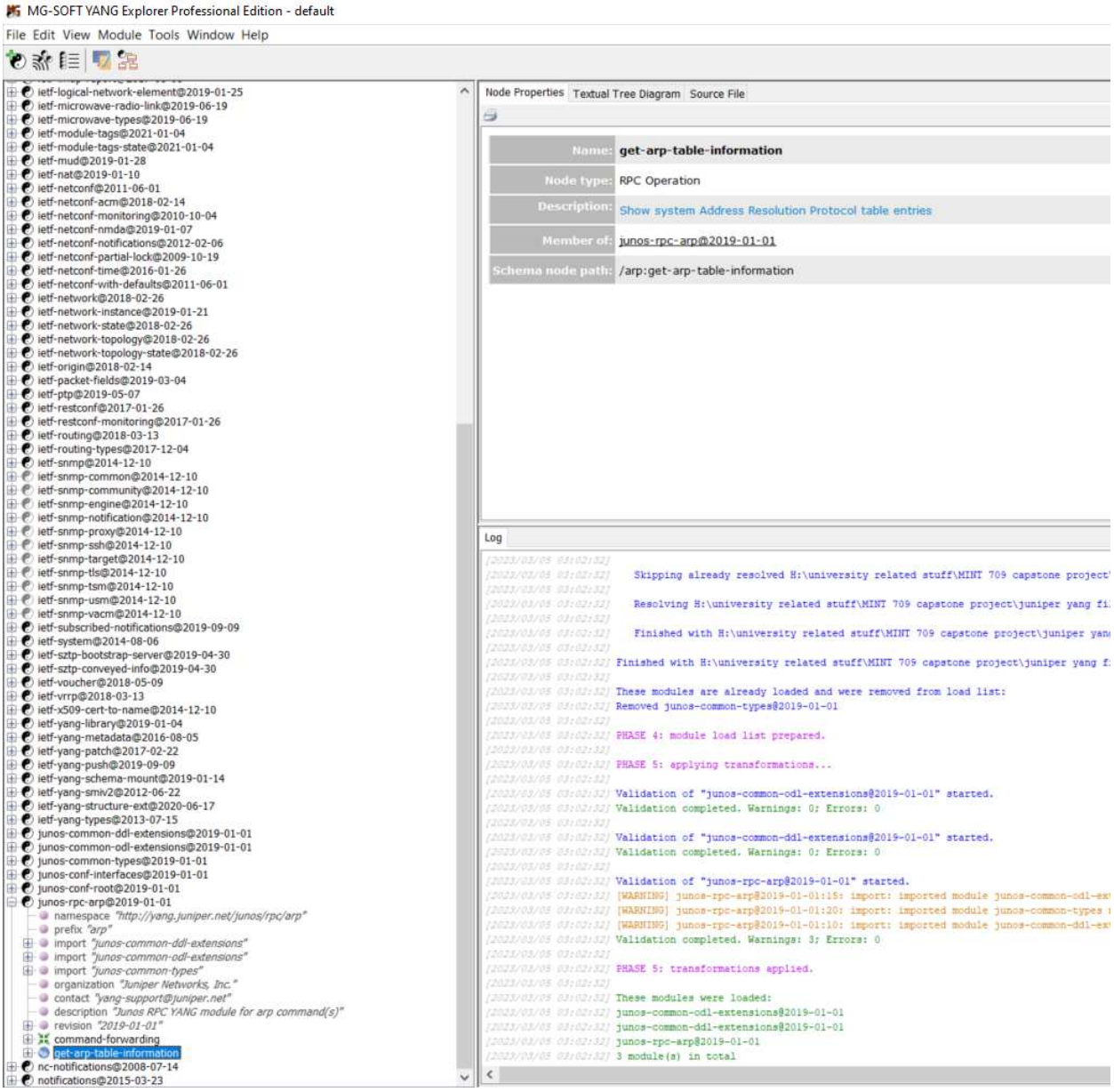



*Figure 4.2.1.7 "get-arp-table-information" rpc call inside junos-rpc-arp YANG module*

On "source File" tab the structure of this RPC is shown. Under this RPC there are different leaf objects such as hostname, interface and so on. Figure 4.2.1.8 shows that YANG module structure (leaf objects) corresponds directly to the CLI options for "**show arp**" command.

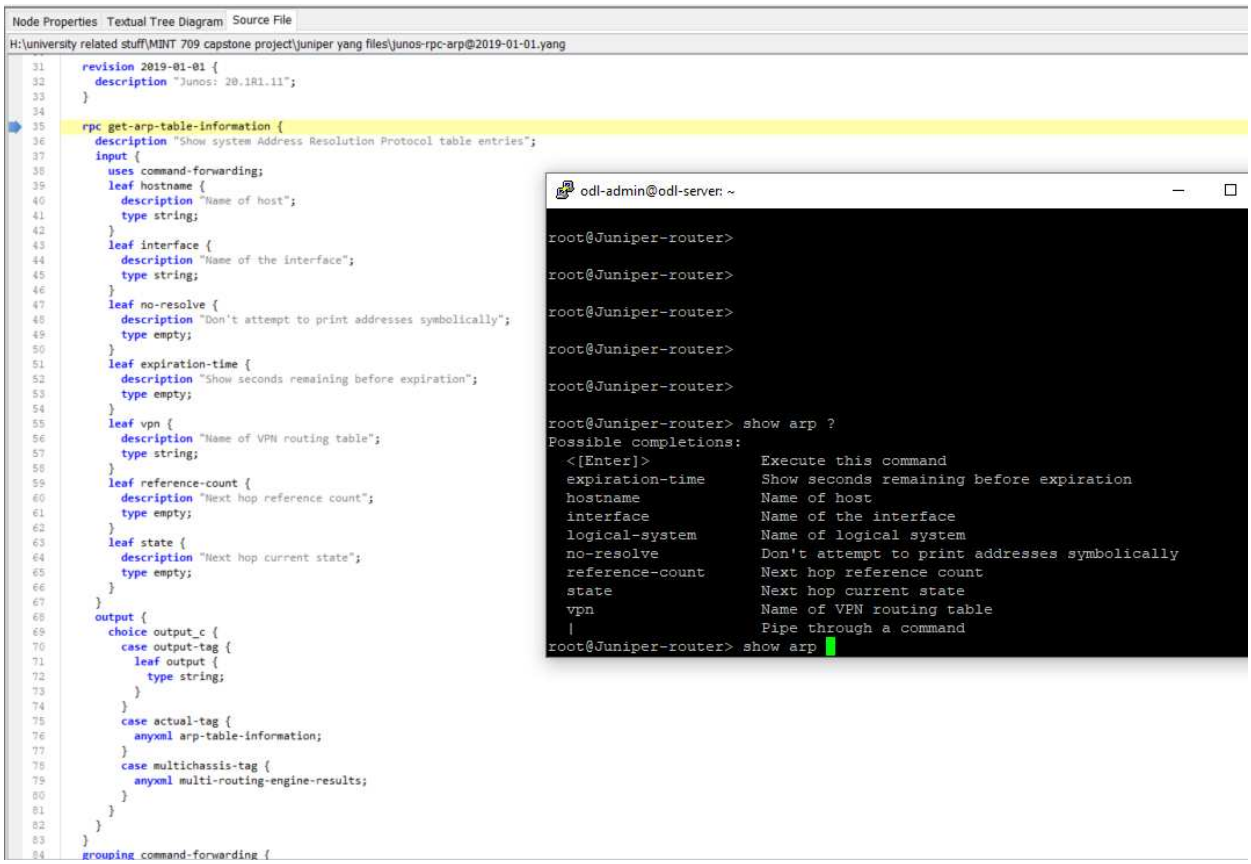



*Figure 4.2.1.8 YANG module structure corresponds to CLI structure.*

So now after learning how NETCONF operation command work and how YANG modules are defined and match to cli request. Let's take a look at few configuration examples.

## 4.2.2 Change interface IP

**Step1:** In order to change interface configuration with NETCONF, first we need to find the YANG data model. We can achieve this goal by either using Juniper GitHub YANG repository or by checking the desired interface configuration from CLI and pipe the output with "**display xml**" at the end. This is shown in the below example. Then copy the entire output surrounded by the **<configuration> … </configuration>** XML tags which is highlighted in the below example.

```
root@Juniper-R1# run show configuration interfaces ge0/0/0 | display xml
<rpc-reply xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
    <configuration Junos:commit-seconds="1676165895" Junos:commit-localtime="2023-02-12
01:38:15 UTC" Junos:commit-user="root">
        <interfaces>
            <interface>
                <name>ge-0/0/0</name>
                <unit>
                    <name>0</name>
                    <family>
```

```
                    <inet>
                      <address>
                        <name>10.11.12.13/24</name>
                      </address>
                    </inet>
                  </family>
                </unit>
              </interface>
            </interfaces>
          </configuration>
          <cli>
            <banner>[edit]</banner>
          </cli>
        </rpc-reply>

        [edit]
        root@Juniper-R1#
```

**Step2:** Send the EDIT-CONFIG RPC

Now, using the block below, we'll send a Remote Procedure Call to change the candidate settings. By simply changing the child data inside the configuration element, this request may be applied to any configuration.

```
<rpc>
 <edit-config>
  <target>
   <candidate />
  </target>
  <default-operation>merge</default-operation>
            <config>
                    <configuration Junos:commit-seconds="1675003066" Junos:commit-
                    localtime="2023-01-29 14:37:46 UTC" Junos:commit-user="root">
                                <interfaces>
                                        <interface>
                                                <name>ge-0/0/0</name>
                                                <unit>
                                                        <name>0</name>
                                                        <family>
                                                                <inet>
                                                                        <address>
                                                                                <name>20.21.22.23/24</
                                                                                name>
                                                                        </address>
                                                                </inet>
                                                        </family>
                                                </unit>
                                        </interface>
```

```
                    </interfaces>
                </configuration>
            </config>
        </edit-config>
    </rpc>
]]>]]>
```

**NOTE:**
"Devices running Junos OS have the following edit configuration modes:
**Merge**: The device merges new configuration data into the existing configuration data. This is the default.

**Replace:** The device replaces existing configuration data with the new configuration data.

**None:** The device does not change the existing configuration unless the new configuration element includes an operation attribute." [28]

In this request we have used merge mode which means this IP address 20.21.22.23/24 will be added to the list of secondary IP addresses that are currently assigned to interface ge-0/0/0 unit 0.

Also note that on Junos operating system the only acceptable value for the **<target>** element is **<candidate/>**. [29]

After sending above RPC call, we should receive the following message which indicate whether configuration was changed successfully or not?

```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
<nc:ok/>
</nc:rpc-reply>
]]>]]>
```

To check the candidate configurations that are still not committed to current running configuration, we can use the following command.

```
<rpc>
  <get-configuration compare="rollback" database="candidate" />
</rpc>
]]>]]>
```

The RPC response would basically show an XML representation of the new configuration that still need to be committed on the Juniper device.
To apply (commit) new changes to the Juniper device we can use any of the following commands. Which respectively equal to "commit confirmed" and "commit" CLI commands.

Commit confirmed RPC request

```
<rpc>
 <commit>
  <confirmed/>
 </commit>
</rpc>
]]>]]>
```

Commit the RPC request

```
<rpc>
 <commit>
 </commit>
</rpc>
]]>]]>
```

### 4.2.3 Change Hostname

Steps for changing the hostname will be exactly the same, first we need to find the equivalent XML rpc request, then send it enclosed with NETCONF operation tags and at the final step commit the new configuration.

**Step1:** find equivalent XML RPC request

```
root@Juniper-R1# run show configuration system host-name | display xml
<rpc-reply xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
    <configuration Junos:commit-seconds="1676184964" Junos:commit-localtime="2023-02-
12 06:56:04 UTC" Junos:commit-user="root">
        <system>
            <host-name>Juniper-R1</host-name>
        </system>
    </configuration>
    <cli>
        <banner>[edit]</banner>
    </cli>
</rpc-reply>

[edit]
root@Juniper-R1#
```

**Step2:** Change the hostname to "Juniper-router"

```
<rpc>
 <edit-config>
  <target>
   <candidate />
  </target>
  <default-operation>merge</default-operation>
            <config>
                  <configuration Junos:commit-seconds="1675003066" Junos:commit-
                  localtime="2023-01-29 14:37:46 UTC" Junos:commit-user="root">
                        <system>
                              <host-name>Juniper-router</host-name>
                        </system>
                        </configuration>
            </config>
</edit-config>
</rpc>
]]>]]>
```

NETCONF Server response which indicate request was successful.

```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
<nc:ok/>
</nc:rpc-reply>
]]>]]>
```

Compare rollback and committed configuration:

```
<rpc>
 <get-configuration compare="rollback" database="candidate" />
</rpc>
]]>]]>
```

NETCONF server response:

```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
<configuration xmlns="http://xml.juniper.net/xnm/1.1/xnm"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:yang="urn:ietf:params:xml:ns:yang:1">
  <system>
    <host-name nc:operation="delete"/>
    <host-name nc:operation="create">Juniper-R</host-name>
  </system>
</configuration>
```

```
</nc:rpc-reply>
]]>]]>
```

Checking request that is waiting to be committed from command line

```
root@Juniper-R1# show | compare
[edit system]
-  host-name Juniper-R1;
+  host-name Juniper-R;

[edit]
root@Juniper-R1#
```

**Step3:** Committing the new configuration with confirm

```
<rpc>
 <commit>
  <confirmed/>
 </commit>
</rpc>
]]>]]>
```

NETCONF server response:

```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
<nc:ok/>
</nc:rpc-reply>
]]>]]>
```

Checking result from command line

```
root@Juniper-R#

# commit confirmed will be rolled back in 10 minutes
[edit]
root@Juniper-R# run show configuration system host-name
host-name Juniper-R;

# commit confirmed will be rolled back in 9 minutes
[edit]
root@Juniper-R#
```

### 4.2.4 Change static routes

Like all the previous NETCONF configuration requests, this task also have the same process. In this task we will add new static route 5.5.5.5 to the routing table of Juniper (NETCONF server).

**Step1:** find equivalent XML RPC request

```
root@Juniper-R1# run show configuration routing-options static | display xml
<rpc-reply xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
  <configuration Junos:commit-seconds="1676165895" Junos:commit-localtime="2023-02-
12 01:38:15 UTC" Junos:commit-user="root">
        <routing-options>
          <static>
            <route>
              <name>2.2.2.2/32</name>
              <next-hop>192.168.1.1</next-hop>
            </route>
            <route>
              <name>3.3.3.3/32</name>
              <next-hop>192.168.1.1</next-hop>
            </route>
            <route>
              <name>4.4.4.4/32</name>
              <next-hop>192.168.1.1</next-hop>
            </route>
          </static>
        </routing-options>
  </configuration>
  <cli>
    <banner>[edit]</banner>
  </cli>
</rpc-reply>

[edit]
root@Juniper-R1#
```

**Step2:** Add 5.5.5.5 route to the static routing table:

```
<rpc>
 <edit-config>
  <target>
   <candidate />
  </target>
  <default-operation>merge</default-operation>
              <config>
```

```
                    <configuration Junos:commit-seconds="1675003066" Junos:commit-
                    localtime="2023-01-29 14:37:46 UTC" Junos:commit-user="root">
                        <routing-options>
                            <static>
                                <route>
                                    <name>5.5.5.5/32</name>
                                    <next-hop>192.168.1.1</next-hop>
                                </route>
                            </static>
                        </routing-options>
                    </configuration>
                </config>
</edit-config>
</rpc>
]]>]]>
```

Compare rollback and committed configuration

```
<rpc>
  <get-configuration compare="rollback" database="candidate" />
</rpc>
]]>]]>
```

Response from the NETCONF server (Juniper device)

```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
<configuration xmlns="http://xml.juniper.net/xnm/1.1/xnm"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:yang="urn:ietf:params:xml:ns:yang:1">
   <routing-options>
      <static>
         <route yang:insert="after"  yang:key="[ name='4.4.4.4/32' ]" nc:operation="create">
            <name>5.5.5.5/32</name>
            <next-hop>192.168.1.1</next-hop>
         </route>
      </static>
   </routing-options>
</configuration>
</nc:rpc-reply>
]]>]]>
```

**Step3:** Commit the new configuration with confirm

```
<rpc>
 <commit>
 </commit>
</rpc>
]]>]]>
```

Response from NETCONF server:

```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
<nc:ok/>
</nc:rpc-reply>
]]>]]>
```

Checking route table after commit:

```
root@Juniper-router# run show configuration routing-options static | display xml
<rpc-reply xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
  <configuration Junos:commit-seconds="1676184355" Junos:commit-localtime="2023-02-
12 06:45:55 UTC" Junos:commit-user="root">
      <routing-options>
        <static>
          <route>
            <name>2.2.2.2/32</name>
            <next-hop>192.168.1.1</next-hop>
          </route>
          <route>
            <name>3.3.3.3/32</name>
            <next-hop>192.168.1.1</next-hop>
          </route>
          <route>
            <name>4.4.4.4/32</name>
            <next-hop>192.168.1.1</next-hop>
          </route>
          <route>
            <name>5.5.5.5/32</name>
            <next-hop>192.168.1.1</next-hop>
          </route>
        </static>
      </routing-options>
  </configuration>
  <cli>
    <banner># commit confirmed will be rolled back in 10 minutes
      [edit]</banner>
    </cli>
  </rpc-reply>
```

## 4.3 Juniper Automation using OpenDaylight and NETCONF

So far, we have covered the concept of NETCONF and SDN and also shown different approaches that we can use to automate Junos operating systems such as REST API and NETCONF.
The final part will cover how to use OpenDaylight SDN controller to manage Juniper device via NETCONF.

In OpenDaylight we can use RESTCONF protocol to send configuration request to the remote NETCONF devices. According to OpenDaylight documents:

"The NETCONF southbound plugin is capable of connecting to remote NETCONF devices and exposing their configuration/operational datastores, RPCs and notifications as MD-SAL mount points. These mount points allow applications and remote users (over RESTCONF) to interact with the mounted devices." [30]

That means by using MD-SAL mountpoints, OpenDaylight can connect to NETCONF servers (Devices), read and learn the YANG modules supported by the remote NETCONF devices then store them in the model cache directory to be used by other Applications. Then OpenDaylight user can use RESTCONF protocol to interact with NETCONF protocol. All the magic is happening in background in OpenDaylight core. Figure 4.3.1 shows how a new device is added to the MD-SAL node inventory and model cache to be used later through the northbound RESTCONF interface. In the below picture, when Juniper1 is added to MD-SAL Node inventory, all the related YANG modules and schemas advertised from device are added to MD-SAL Model cache. Then if we want to add Juniper2 to the Node inventory of MD-SAL because all the related modules and schemas were already added to the Model cache, this step will be skipped. But, adding the Cisco XR router to the Node inventory again will require all the related modules and schemas to be added to the Model Cache.

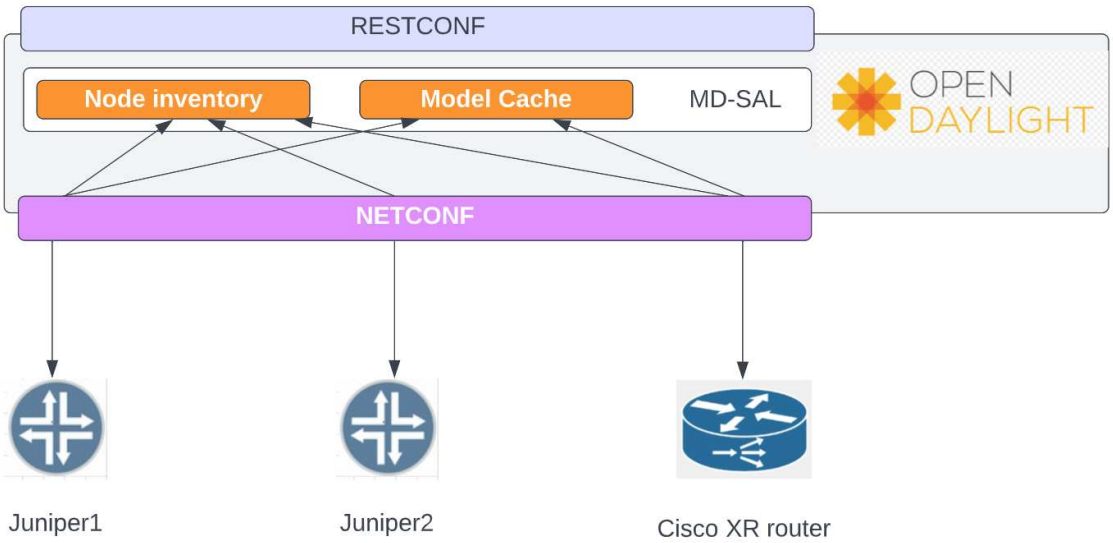



*Figure 4.3.1 How new NETCONF device is added to MD-SAL - idea from source: cisco DEVNET 1119 – Using OpenDaylight (https://www.youtube.com/watch?v=rAm48gVv8_A&ab_channel=CiscoDevNet)*

The OpenDaylight user will interact with MD-SAL mountpoint as shown in the picture below. And translating RESTCONF request to NETCONF which should be sent to NETCONF device will happen through MD-SAL NETCONF mountpoints.
Later we will show how it will work in action.

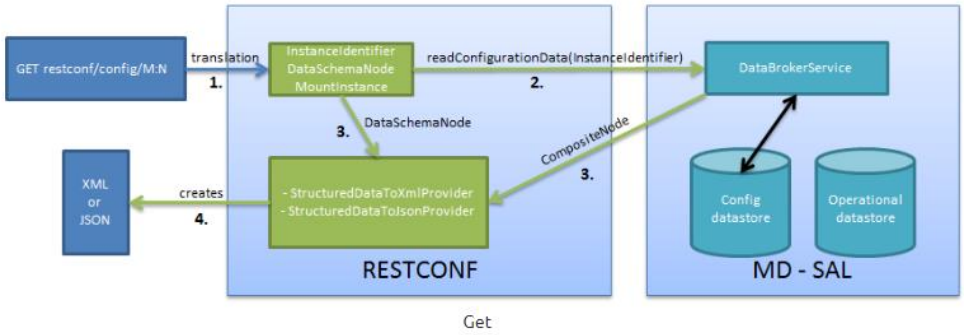



*Figure 4.3.2 RESTCONF transaction flow in OpenDaylight source: https://docs.opendaylight.org/en/stable-boron/user-guide/netconf-user-guide.html#netconf-connector-configuration-with-md-sal*

Before we see how NETCONF automation work in action in OpenDaylight, let's first look at how our lab environment is set up. Below figure shows lab environment structure.

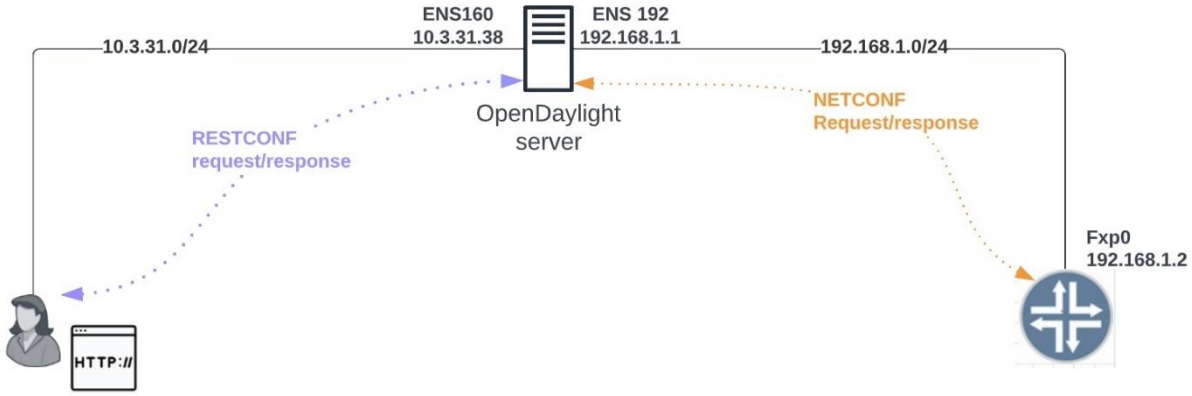



*Figure 4.3.3 OpenDaylight NETCONF lab diagram*

## 4.3.1 OpenDaylight southbound NETCONF configuration:

Steps to configure OpenDaylight NETCONF southbound and RESTCONF

1- Make sure OpenDaylight is running.
2- Activate NETCONF southbound.

Installing "**odl-netconf-connector-all**" Karaf feature will activate NETCONF southbound on OpenDaylight server.

3- Install other prerequisite packages

Install "**odl-restconf-nb-bierman02**" Karaf feature
Install "**odl-resconf**" Karaf feature

Check the installation was successful and all feature are started with the following commands:

feature:list | grep netconf | grep Started
feature:list | grep restconf | grep Started

```
opendaylight-user@root>feature:list | grep netconf  | grep Started
odl-netconf-connector-all                         (0x 3.0.8          | x        | Started     | odl-netconf-connector-all
                | OpenDaylight :: Netconf Connector :: All
odl-netconf-topology                              (0x 3.0.8          |          | Started     | odl-netconf-topology
                | OpenDaylight :: Netconf Topology :: Netconf Conne
odl-restconf-common                               (0x 3.0.8          |          | Started     | odl-netconf-3.0.8
                | OpenDaylight :: Restconf :: Common
odl-netconf-netty-util                            (0x 3.0.8          |          | Started     | odl-netconf-netty-util-3.0.8
                | OpenDaylight :: Netconf :: Netty Util
odl-netconf-mapping-api                           (0x 3.0.8          |          | Started     | odl-netconf-mapping-api
                | OpenDaylight :: Netconf :: Mapping API
odl-netconf-connector                             (0x 3.0.8          |          | Started     | odl-netconf-3.0.8
                | OpenDaylight :: Netconf Connector :: Netconf Conn
odl-netconf-client                                (0x 3.0.8          |          | Started     | odl-netconf-client-3.0.8
                | odl-netconf-client
odl-netconf-api                                   (0x 3.0.8          |          | Started     | odl-netconf-3.0.8
                | OpenDaylight :: Netconf :: API
odl-netconf-notifications-api                     (0x 3.0.8          |          | Started     | odl-netconf-notifications-api
                | OpenDaylight :: Netconf :: Notification :: Api
odl-netconf-ssh                                   (0x 3.0.8          |          | Started     | odl-netconf-ssh
                | OpenDaylight :: Netconf Connector :: SSH
odl-netconf-impl                                  (0x 3.0.8          |          | Started     | odl-netconf-impl-3.0.8
                | OpenDaylight :: Netconf :: Impl
odl-netconf-util                                  (0x 3.0.8          |          | Started     | odl-netconf-3.0.8
                | odl-netconf-util
odl-netconf-notifications-impl                    (0x 3.0.8          |          | Started     | odl-netconf-notifications-impl-3.0.8
                | OpenDaylight :: Netconf :: Monitoring :: Impl
odl-netconf-callhome-ssh                          (0x 3.0.8          |          | Started     | odl-netconf-callhome-ssh
                | OpenDaylight :: Netconf Connector :: Netconf Call
odl-aaa-netconf-plugin                            (0x 3.0.8          |          | Started     | odl-aaa-netconf-plugin
                | OpenDaylight :: AAA :: ODL NETCONF Plugin
odl-netconf-tcp                                   (0x 3.0.8          |          | Started     | odl-netconf-tcp
                | OpenDaylight :: Netconf Connector :: TCP
opendaylight-user@root>
opendaylight-user@root>
opendaylight-user@root>feature:list | grep restconf  | grep Started
odl-restconf                                      (0x 3.0.8          | x        | Started     | odl-restconf
                | OpenDaylight :: Restconf
odl-restconf-nb-bierman02                         (0x 3.0.8          | x        | Started     | odl-restconf-nb-bierman02
                | OpenDaylight :: Restconf :: NB :: bierman02
odl-restconf-nb-rfc8040                           (0x 3.0.8          |          | Started     | odl-restconf-nb-rfc8040-3.0.8
                | OpenDaylight :: Restconf :: NB :: RFC8040
odl-restconf-common                               (0x 3.0.8          |          | Started     | odl-netconf-3.0.8
                | OpenDaylight :: Restconf :: Common
opendaylight-user@root>
```

*Figure 4.3.1.1 check all necessary netconf and restonf services are installed and running*

4- Spawn a NETCONF-connector (network device)

There are two approaches to create a new NETCONF-connector. Either use PUT or POST method.

PUT http://10.3.31.38:8181/RESTCONF/config/network-topology:network-topology/topology/topology-netconf/node/junipervmx1

OR

POST http://10.3.31.38:8181/RESTCONF/config/network-topology:network-topology/topology/topology-netconf

The Payload content will be the same for both methods.

**BODY:**

**Content-type:** application/xml

**Accept:** application/xml

**Authentication:** admin:admin

```
<node xmlns="urn:TBD:params:xml:ns:yang:network-topology">
 <node-id>junipervmx1</node-id>
 <host xmlns="urn:opendaylight:netconf-node-topology">192.168.1.2</host>
 <port xmlns="urn:opendaylight:netconf-node-topology">830</port>
 <username xmlns="urn:opendaylight:netconf-node-topology">user</username>
 <password xmlns="urn:opendaylight:netconf-node-topology">password</password>
 <tcp-only xmlns="urn:opendaylight:netconf-node-topology">false</tcp-only>
 <!-- non-
mandatory fields with default values, you can safely remove these if you do not wish to override any of thes
e values-->
 <reconnect-on-changed-schema xmlns="urn:opendaylight:netconf-node-topology">false</reconnect-on-
changed-schema>
 <connection-timeout-millis xmlns="urn:opendaylight:netconf-node-topology">20000</connection-
timeout-millis>
 <max-connection-attempts xmlns="urn:opendaylight:netconf-node-topology">0</max-connection-
attempts>
 <between-attempts-timeout-millis xmlns="urn:opendaylight:netconf-node-topology">2000</between-
attempts-timeout-millis>
 <sleep-factor xmlns="urn:opendaylight:netconf-node-topology">1.5</sleep-factor>
 <!-- keepalive-delay set to 0 turns off keepalives-->
 <keepalive-delay xmlns="urn:opendaylight:netconf-node-topology">120</keepalive-delay>
</node>
```

**Note that in body content:**

**Host** element value will be the IP address of the NETCONF device
**Port** element value will be the NETCONF TCP port number that is enabled on the device
**User** element value will be the user account who can get access to device via NETCONF
Password element value will be the password of the user for NETCONF connection

After sending the request if we receive a 2xx response then it means request was successful and NETCONF-connector is created.

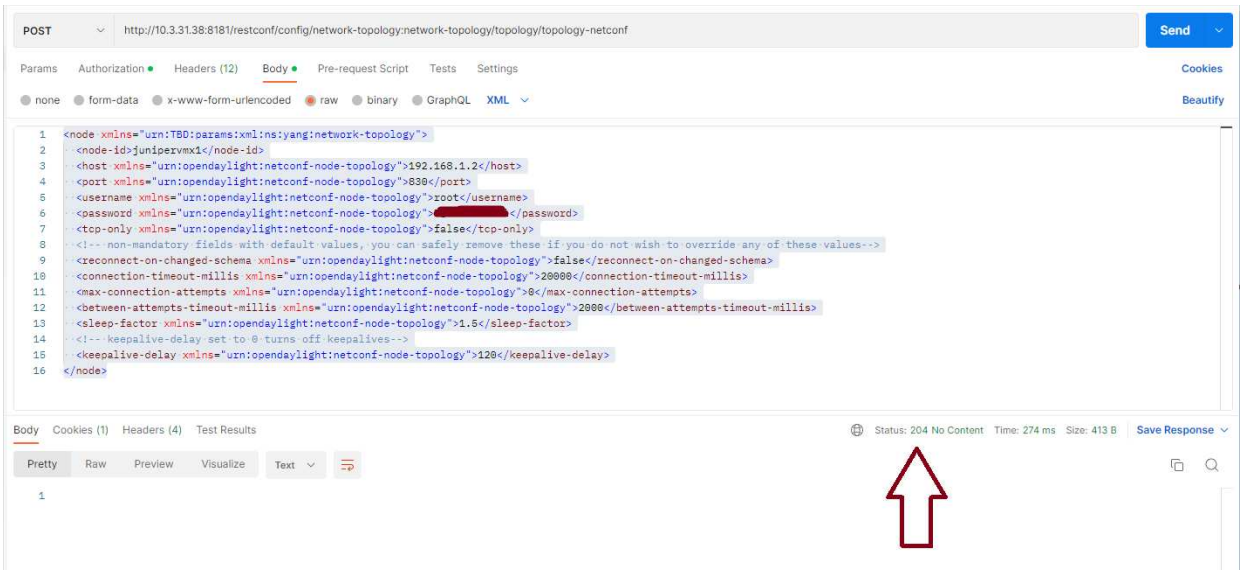



*Figure 4.3.1.2 Successful creation of a new NETCONF Connector*

5- Check NETCONF device capabilities

Using following GET request we can get a list of all NETCONF connectors that are available on OpenDaylight SDN controller and details about each of them. This information includes:

- device connection status
- node id
- list of all capabilities advertised by the NETCONF device.

GET http://10.3.31.38:8181/RESTCONF/operational/network-topology:network-topology/topology/topology-netconf/

The response will show NETCONF device detail with a list of capabilities that device support. In this case we have only one NETCONF connector (junipervmx1). But if we have multiple NETCONF connectors, all of them will be shown in the response of the above request. (Full response body is available at Appendix B section 7.2)


```
{
  "topology": [
    {
      "topology-id": "topology-netconf",
      "node": [
        {
          "node-id": "junipervmx1",
          "netconf-node-topology:connection-status": "connected",
          "netconf-node-topology:port": 830,
          "netconf-node-topology:available-capabilities": {
            "available-capability": [
              {
                "capability": "urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,file",
```

```
                "capability-origin": "device-advertised"
              },
              {
                "capability": "urn:ietf:params:netconf:capability:confirmed-commit:1.0",
                "capability-origin": "device-advertised"
              },
              {
                "capability": "urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0",
                "capability-origin": "device-advertised"
              },
              .
              .
              Full response body available at Appendix B section 7.2
              .
              .
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/interfaces?revision=2019-01-01)Junos-
rpc-interfaces"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/ldp?revision=2019-01-01)Junos-rpc-
ldp"
              }
            ]
          },
          "netconf-node-topology:host": "192.168.1.2"
        }
      ]
    }
  ]
}
```


**NOTE:** by default, responses are shown in json format, but it is possible to receive the output in XML format. To see the response in XML format, change the Accept header default value from "***/***" to "**application/xml**"

### 4.3.2 Configure NETCONF device with RESTCONF call from OpenDaylight

Now that we have created our NETCONF connector, the next step is to fetch the current configuration from NETCONF device. In order to get running configuration on the NETCONF device again we will use the GET command but this time, at the end of the URL from previous step we will need to add "**/node/<*node id name*>/yang-ext:mount**" in which node id name is the name of the desired NETCONF-connector device that we want to read data from it. In this example, as we saw in the previous step only one node is defined with the name "junipervmx1", which is used in the following request.

GET http://10.3.31.38:8181/RESTCONF/operational/network-topology:network-topology/topology/topology-netconf/node/junipervmx1/yang-ext:mount/

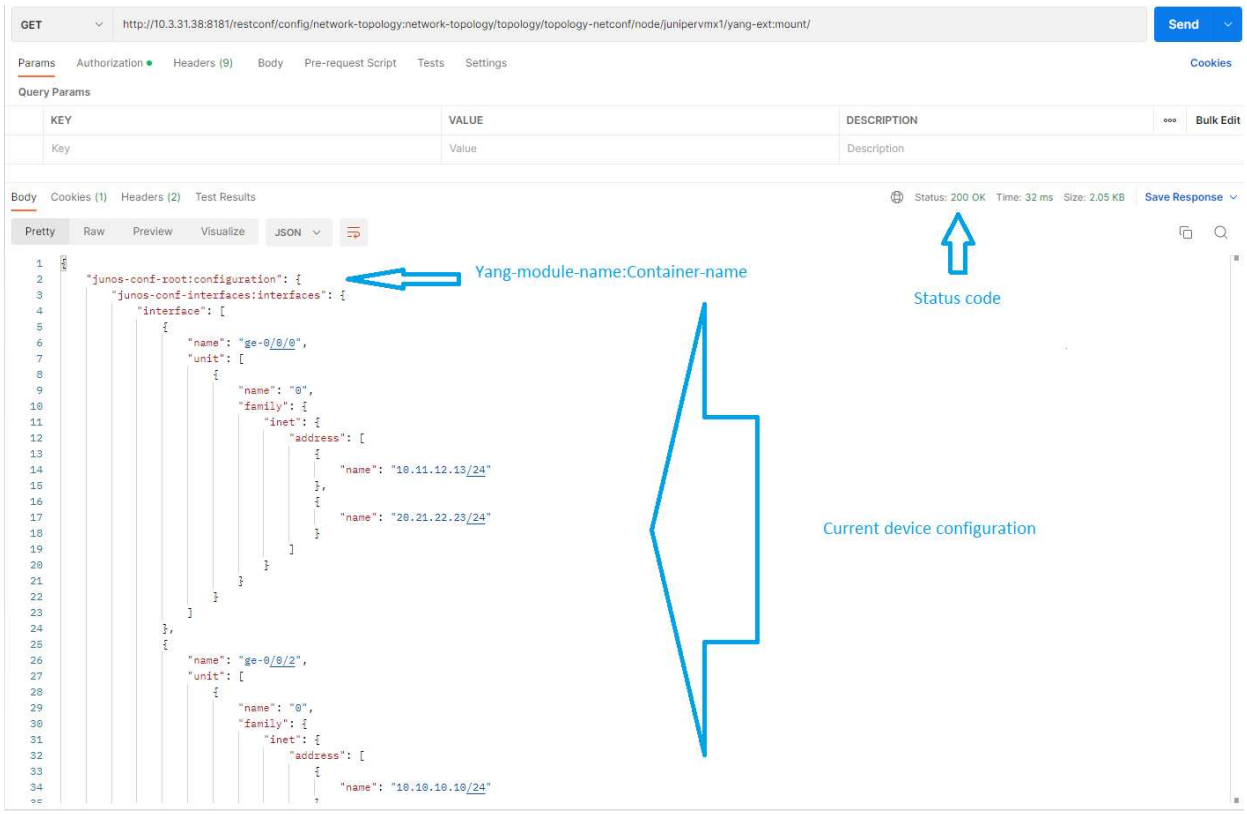



*Figure 4.3.2.1 Get configuration of a NETCONF node with RESTCONF request*

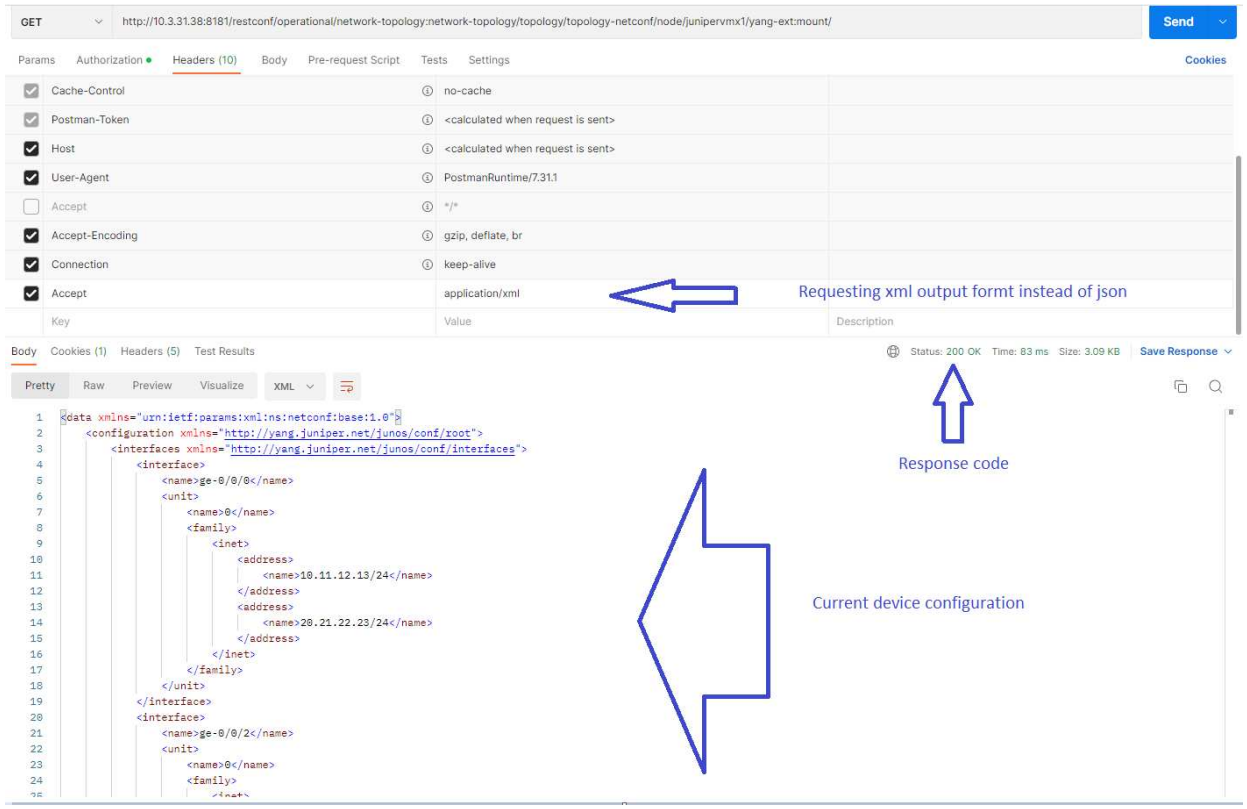



*Figure 4.3.2.2 Get configuration of a NETCONF node with RESTCONF request and XML format instead of JSON*

How to change interface IP address with RESTCONF request:

Looking back at the response from the previous step, on line 2 main YANG module name and its container are represented in the format of <YANG module name>:<Container>. In this step, in order to edit the interface configuration RESTCONF method and URL path format are as follows:

PUT http://10.3.31.38:8181/RESTCONF/config/network-topology:network-topology/topology/topology-netconf/node/junipervmx1/yang-ext:mount/Junos-conf-root:configuration/

For body of this request, we will copy the whole response body from previous step and paste it in the body section of this new request. Then change the element value that we want and send the request. For example, in this case we will add the IP address of 30.31.32.33/24 as a new secondary IP address under interface ge0/0/0. all the rest of the body would remain the same. Look at the following picture.

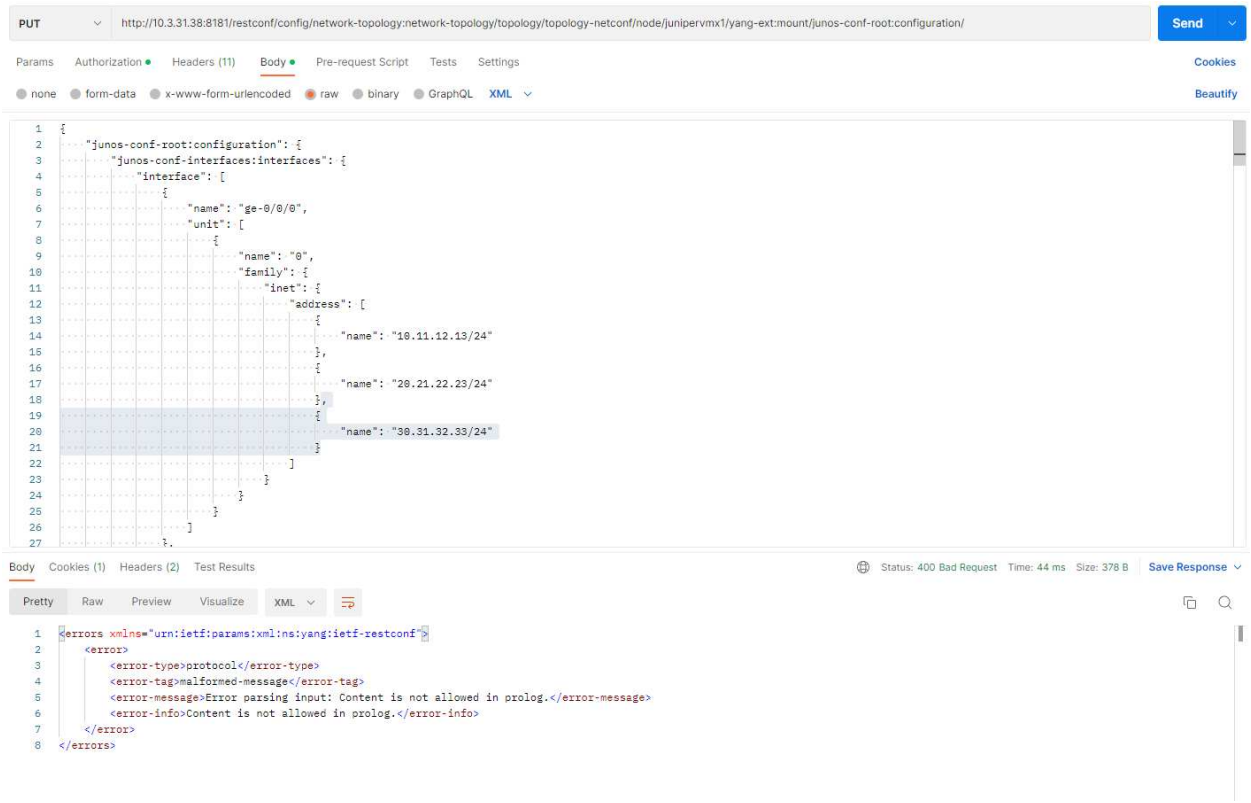



*Figure 4.3.2.3 Change NETCONF device interface configuration with RESTCONF request*

Only the highlighted part was added to the response received from the previous step. But now the response code is 400 which means something is wrong. It indicates content type is not allowed. Which make sense we are sending **JSON** format body but in the header, we have selected **XML** as content type. So that is the reason why sending the request failed. Let's try it one more time but this time change the content type header to "**JSON**"

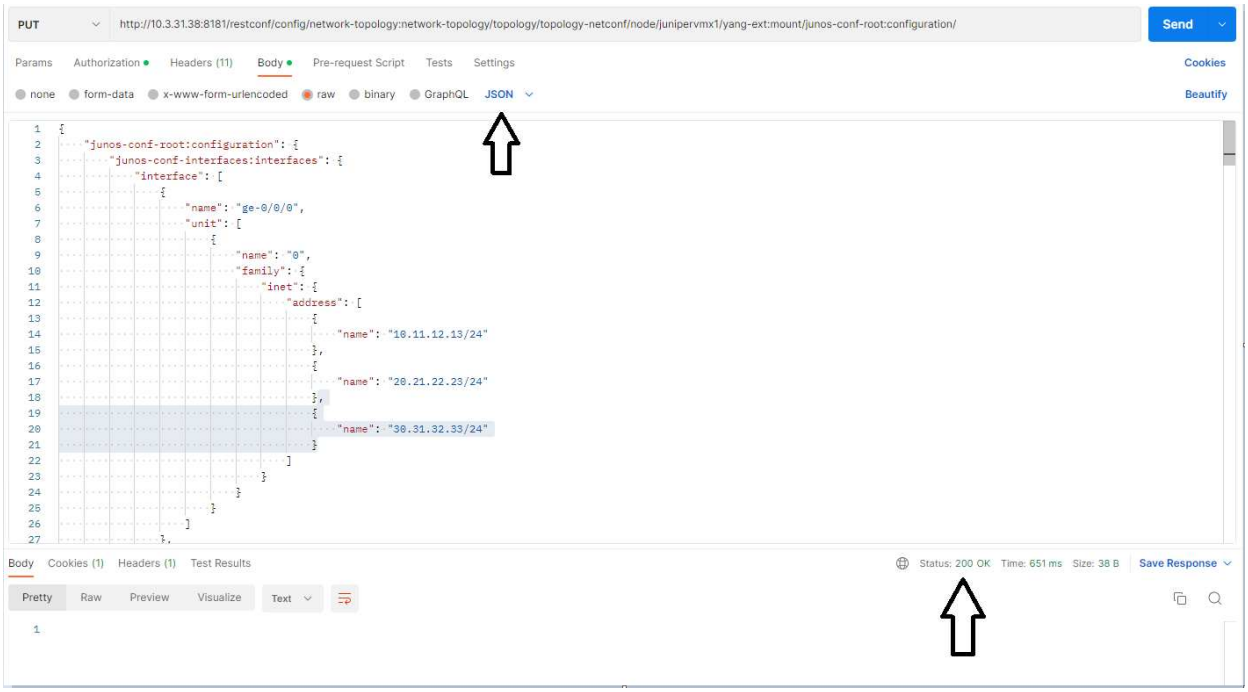



*Figure 4.3.2.4 Sending the request with proper body format will return success status*

Since the response code is 200. that means the configuration change was successfully applied to the NETCONF device. Let's login to the device and make sure new change is applied on the system.

```
root@Juniper-router>

root@Juniper-router> show configuration interfaces ge0/0/0
unit 0 {
    family inet {
        address 10.11.12.13/24;
        address 20.21.22.23/24;
        address 30.31.32.33/24;
    }
}

root@Juniper-router>
```

To add/delete or change any configuration using RESTCONF, we can use the same approach.

## Change HOSTNAME

For the next example, let's change the hostname on Juniper device.
With the same method used in previous step, let's first read the current configuration from the NETCONF device.

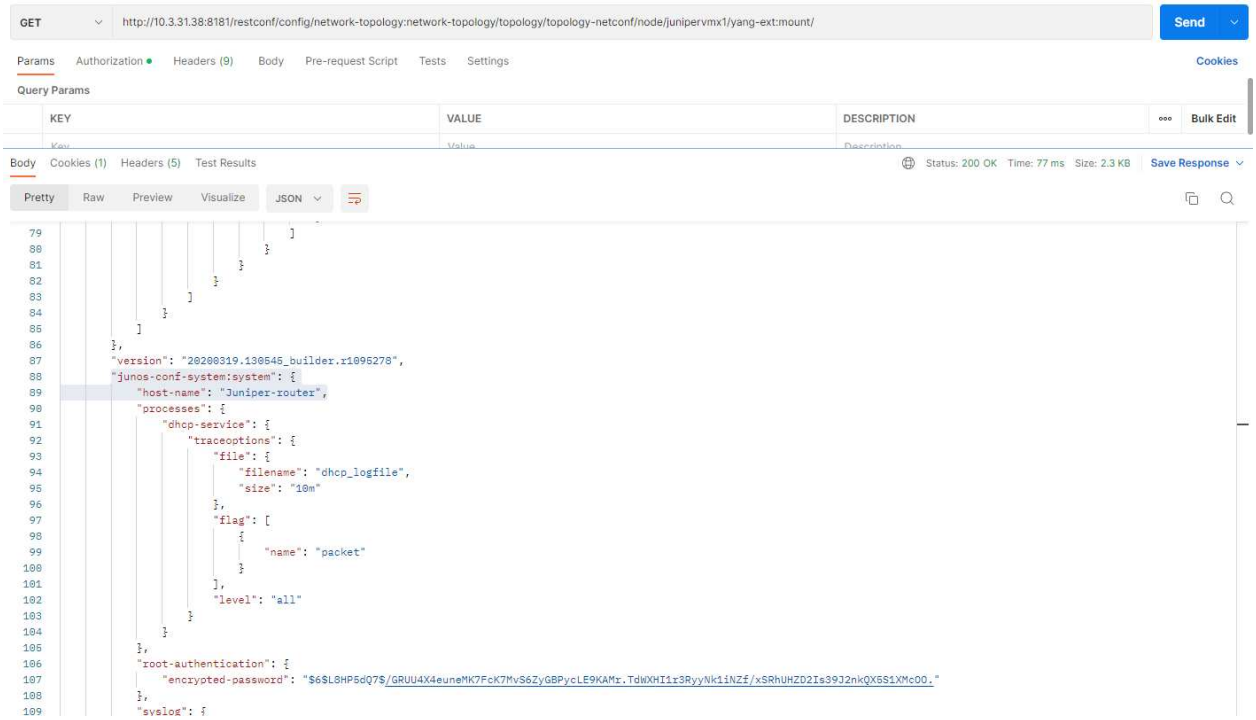



*Figure 4.3.2.5 Reading the current configuration from NETCONF device*

For changing the hostname, the same method and URL path used in the previous step are going to be used in this step too.
But now, the question is can we do it in another way?
The answer is yes. If we take a deeper look at the structure of response from previous request, it is evident that each part of configuration belongs to a specific YANG module and container in it. For example looking at the Figure 4.3.2.5, hostname configuration resides under YANG module "**junos-conf-system"** and "**system"** container (**"Junos-conf-system:system"**). It is possible to add these two YANG modules and their container to the URL path to narrow down the response content to only desired part. It is shown in the below picture.

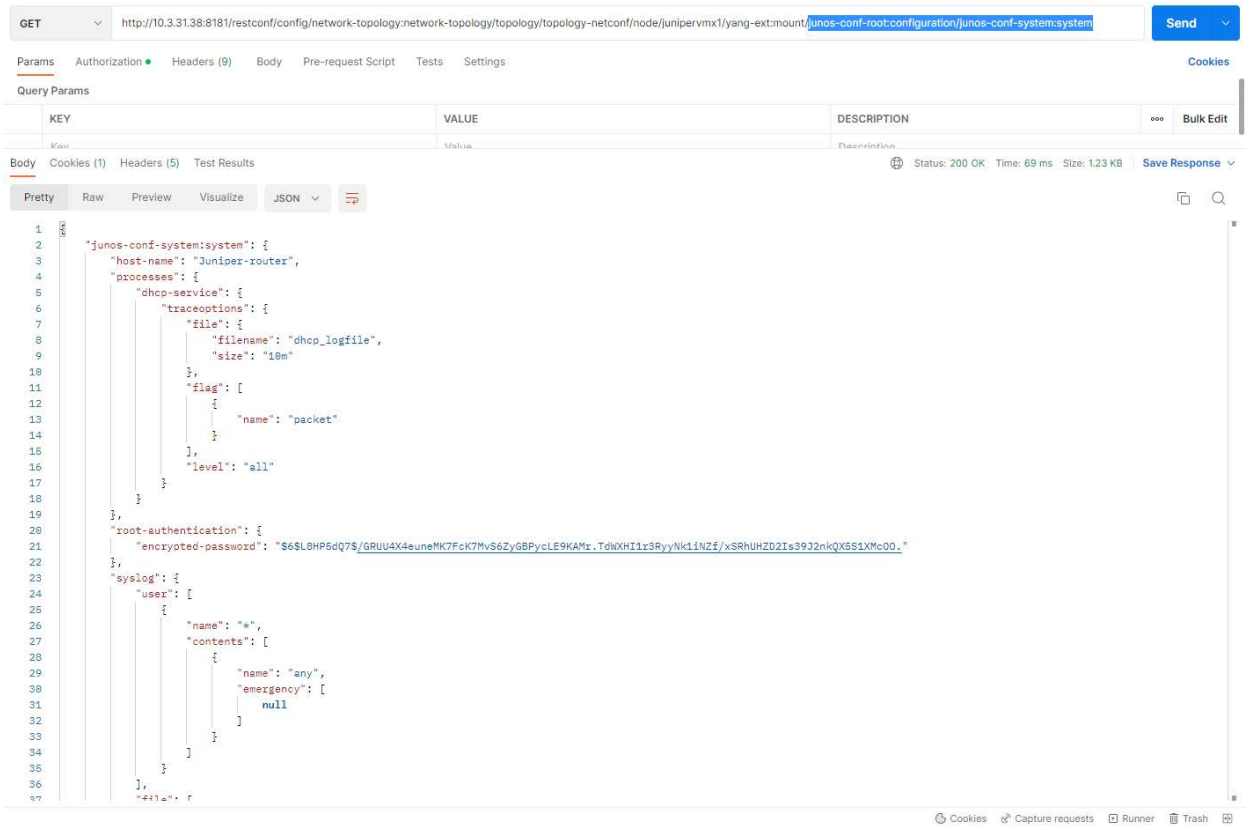



*Figure 4.3.2.6 Getting partial of running configuration of NETCONF device that is only related to system container of junos-conf-system YANG module.*

Now to have a better understanding of how RESTCONF outputs match with the YANG structure of the Junos. In the below picture, we tried to show the side-by-side mapping between YANG modoule "**junos-conf-system**" hostname leaf node and part of the output from the above picture which is related to hostname in the MG-SOFT YANG Explorer application.

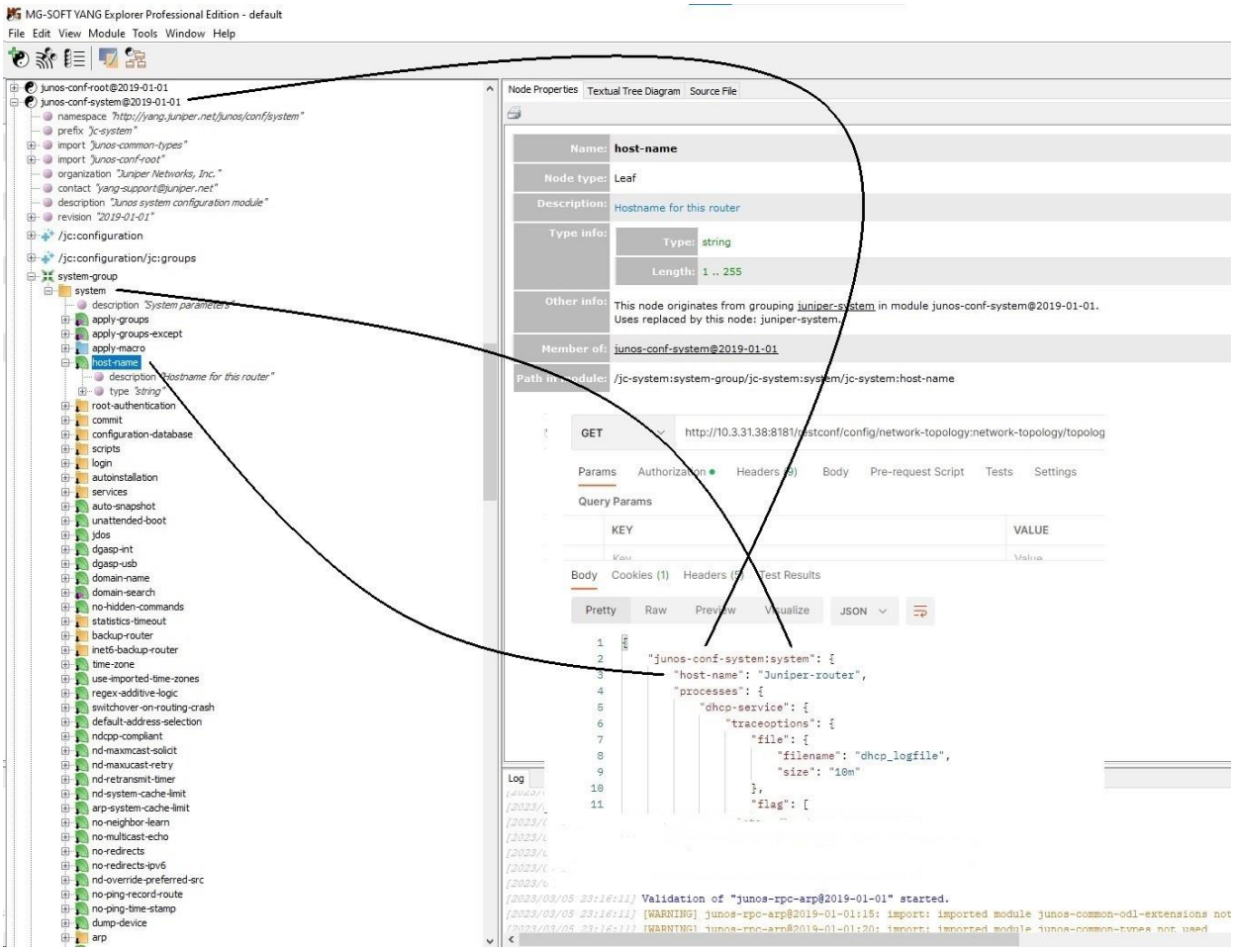



*Figure 4.3.2.7 Mapping JSON body content of a RESTCONF request with the corresponding container and leaf in the YANG module.*

# 5. Conclusion

In this report we tried to briefly cover what SDN is, what is YANG and NETCONF and how they are used together to help automate network device configuration and monitoring. Then different types of Juniper Automation were covered. We saw how to use OpenDaylight to configure NETCONF Juniper devices via RESTCONF protocol and among all the juniper automation technics that are covered in this report, we believe this is the best one, because of the following reason:

Unlike REST-API or direct NETCONF SSH connection, application developers don't need connection (either REST or SSH) to different devices and all are handled by MD-SAL and NETCONF connectors in the background. Only a change in RESTCONF URL path which is related to node name would be enough to receive NETCONF operation/configuration data from a new device.
Also, OpenDaylight RESTCONF northbound plugin help to receive NETCONF device data in the format of JSON or XML. This will be very useful in term of application developments. Developers don't need to worry about the underlying NETCONF and SSH connection and only deals with RESTCONF API calls.

This project also has different challenges that need to be addressed here:

Since OpenDaylight is an opensource product, the most challenging issue about this project was lack of documentation

1- Lack of document about how to integrate ODL with juniper
2- Lack of document and detailed explanation about how to work with RESTCONF or even how to call the juniper NETCONF device with RESTCONF
3- Most of the data that are available online about ODL integration are at least outdated by a few years and all belongs to older versions of juniper and ODL, which makes it difficult how to align it with current versions.

But in general Software Defined Networking (SDN) is a promising technology that offers significant flexibility, scalability, and cost efficiency benefits. However, a number of disadvantages also need to be considered, including complexity, single points of failure, limited vendor support, security risks, and a skills gap. While SDN may not be suitable for every organization, those willing to invest the time and resources to deploy and manage SDN can realize significant benefits in network management and performance.

# 6. Appendix A: Opendaylight installation

In this part we will go through all the steps of installing OpenDaylight on Ubuntu 22 LTS server edition. The credit of the installation guide, except for some changes and add-ons, goes to the following website: https://john.soban.ski/install-opendaylight-ubuntu-lts-fast.html

Steps:

1. Install the operating system, install and update the required packages
2. Install Java JRE
3. Download the desired version of OpenDaylight
4. Run OpenDaylight

Step1: Install the operating system, install and update the required packages

After installing Ubuntu 22.07 (because of straightforward installation it is not covered in here), first we need to update the operating system, security patches and applications.
We can do it with **APT** package manager.

1.1 execute the following command to get the list of latest available packages.

```
$ sudo apt-get -y update
```

Result will be similar to picture below:

```
odl-admin@odl-server:~$ sudo apt-get -y update
[sudo] password for odl-admin:
Hit:1 http://ca.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://ca.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:3 http://ca.archive.ubuntu.com/ubuntu jammy-backports InRelease [107 kB]
Get:4 http://ca.archive.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Fetched 336 kB in 1s (485 kB/s)
Reading package lists... Done
odl-admin@odl-server:~$
```

1.2 after fetching the latest available package list, then we need to upgrade existing packages to the latest version. Execute the following command

```
$ sudo apt-get -y upgrade
```

We should see a long number of lines in the output indicating packages name that were upgraded.

1.3 To unzip the OpenDaylight package which we will download in step 3, we will need "unzip" package. execute the following command to install it.

```
$ sudo apt-get -y install unzip
```

At the end, it is possible to ask you about which services to restart. Don't change anything and just click on <ok> button to restart default services.

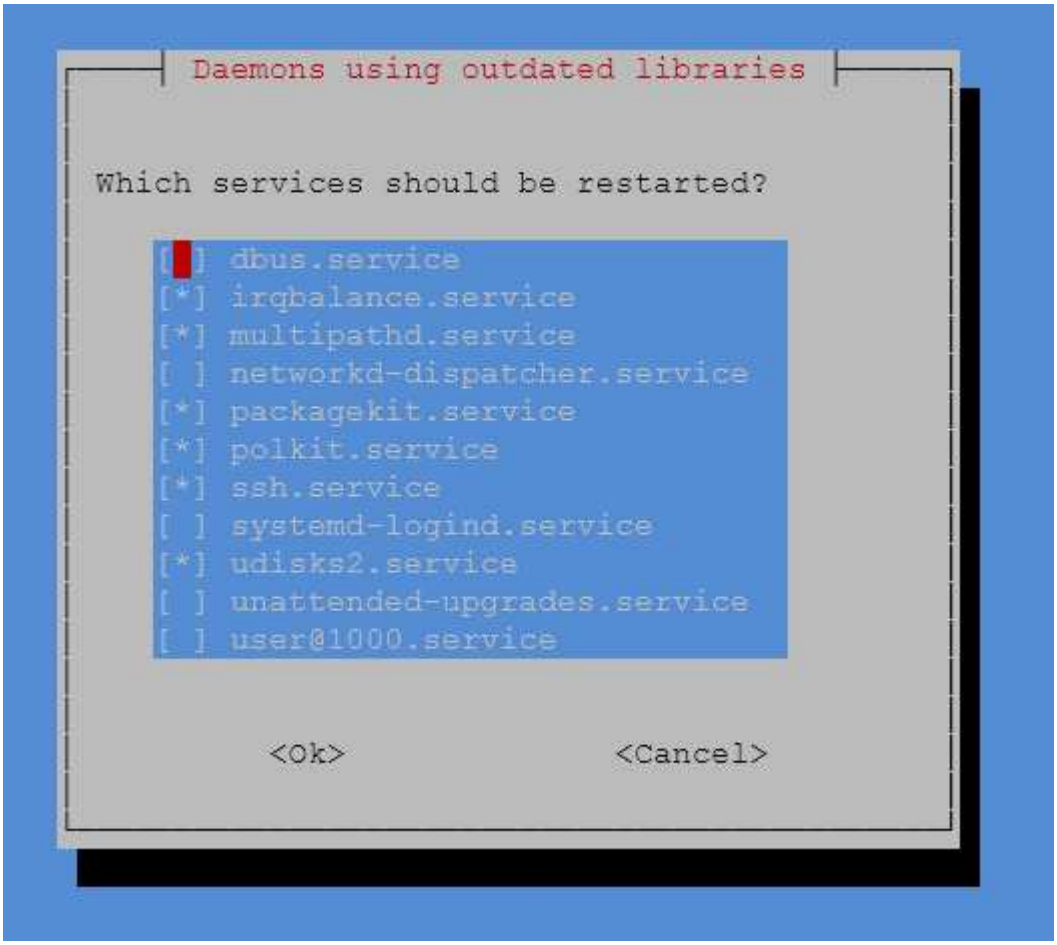



*Figure 6.1 select services to restart after package installation and upgrade*

Step2: Install Java JRE

To be able to run OpenDaylight Sulfur version we will need at least Java 11.

2.1 install Java JRE package

Execute the following command.

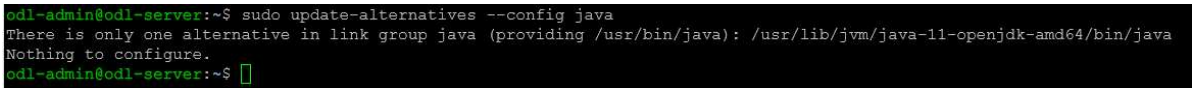
```
~$ sudo apt-get -y install openjdk-11-jre
```

2.2 Change the default Java version

If we have multiple versions of Java installed on our system, then we can set the desired version to be the default using the following command.

```
$ sudo update-alternatives --config java
```

If we only have one version installed on our system and that is version 11, then we don't need to do anything in this step and by using the above command we will receive a message similar to the one in the following picture.




2.3 Set **JAVA_HOME** environment variable

Executing following command, we can retrieve Java executable full path.

```
~$ ls -l /etc/alternatives/java
```

Result will be something like the following picture:

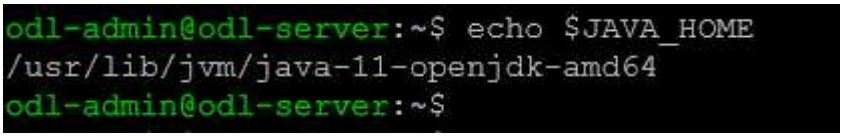

In OpenDaylight, **JAVA_HOME** environment variable should reflect the location the entire JAVA toolset, and not just the JAVA executable. To meet this requirement, we should remove **bin/java** from the path. That will result in **JAVA_HOME** variable to point to the location of JRE.

So now we need to remove "/bin/java" from the end of the last command output and set it as **JAVA_HOME** environmental variable. Since we want to persist this variable's value and don't want it to change in case of reboot or after logout/login, we push the changes to bash resource file as well.

```
~$ echo 'export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64' >> ~/.bashrc
```

To make sure value is set properly, we need to check the output of the following command:

```
echo $JAVA_HOME
```




**NOTE**: Since, Ubuntu read Bash resource file whenever we login to shell, if we can't see any value being set for **JAVA_HOME** we may need to logout and the login back to our shell or use the following command instead of logout/login.

```
$ source ~/.bashrc
```

Step3: Download the desired version of OpenDaylight

According to OpenDaylight.org, currently the stable release is **Sulfur 16.0.**

There are two ways to download and install OpenDaylight. Either we can download the binary code package which then we will need to compile and install in on our system or download the precompiled version.

If we would like to download the binary code to compile it on our own system and then install it, then we can use the following link:

https://docs.opendaylight.org/en/latest/downloads.html

Under Sulfur-SR3 -> Downloads, right click on "OpenDaylight Sulfur Zip" hyperlink and download the zip file. You will need to compile it before running it on your system.

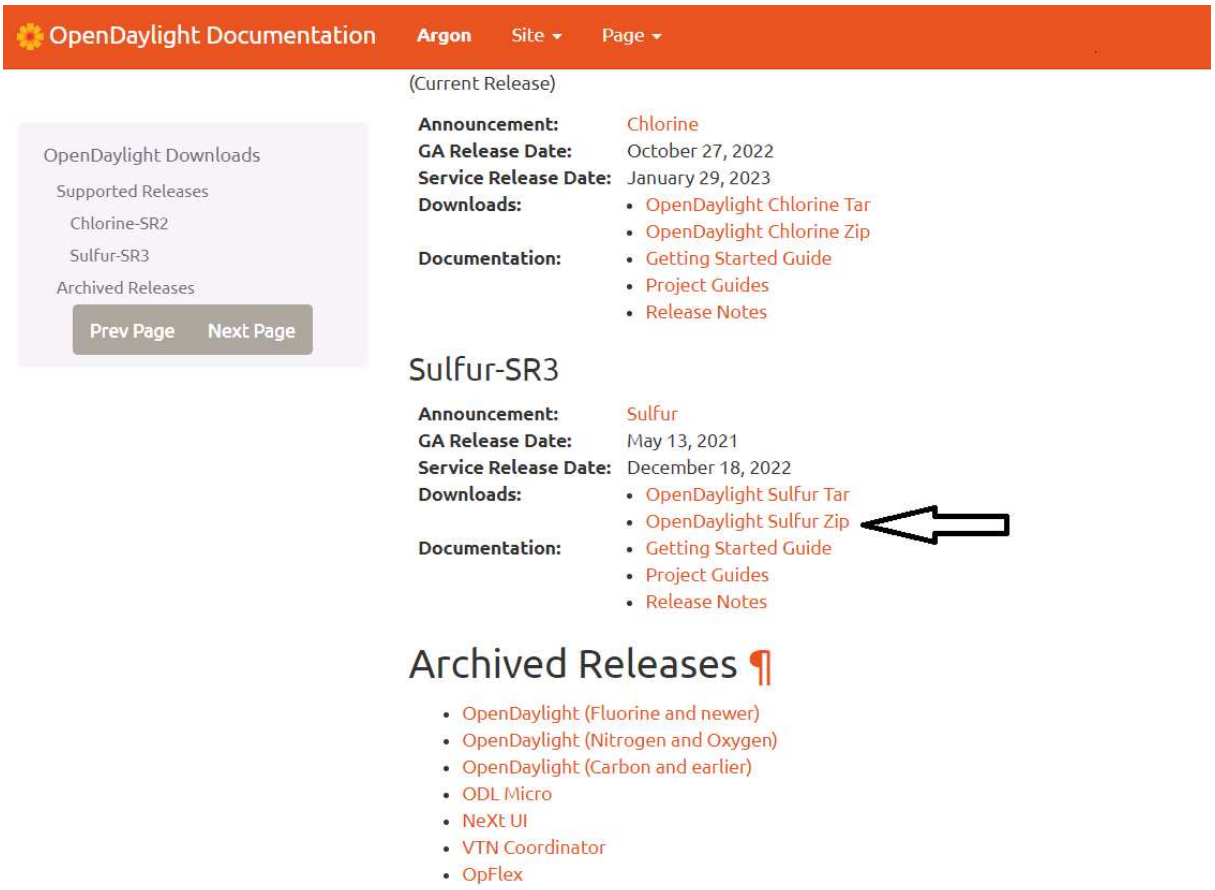



*Figure 6.2 Download zip package of OpenDaylight*

The second easier approach is to download the precompiled version of OpenDaylight package. On the same page, under "Achieved Releases" right-click on "OpenDaylight (Nitrogen and Oxygen)" hyperlink.

On the opened page try to find "0.16.3/" directory which at the moment is the latest version of Sulfur.


On the next page, copy the link of "karaf-0.16.3.zip" file.

| | | |
|---|---|---|
| karaf-0.16.3.tar.gz.asc | Mon Dec 05 01:08:25 UTC 2022 | 455 |
| karaf-0.16.3.tar.gz.asc.md5 | Sun Dec 11 10:26:04 UTC 2022 | 32 |
| karaf-0.16.3.tar.gz.asc.sha1 | Sun Dec 11 10:26:04 UTC 2022 | 40 |
| karaf-0.16.3.tar.gz.asc.sha256 | Sun Dec 11 10:26:04 UTC 2022 | 64 |
| karaf-0.16.3.tar.gz.asc.sha512 | Sun Dec 11 10:26:04 UTC 2022 | 128 |
| karaf-0.16.3.tar.gz.md5 | Mon Dec 05 01:08:21 UTC 2022 | 32 |
| karaf-0.16.3.tar.gz.sha1 | Mon Dec 05 01:08:21 UTC 2022 | 40 |
| karaf-0.16.3.zip | Mon Dec 05 01:08:21 UTC 2022 | 231833138 |
| karaf-0.16.3.zip.asc | Mon Dec 05 01:08:25 UTC 2022 | 455 |
| karaf-0.16.3.zip.asc.md5 | Sun Dec 11 10:26:04 UTC 2022 | 32 |
| karaf-0.16.3.zip.asc.sha1 | Sun Dec 11 10:26:04 UTC 2022 | 40 |

Use the "curl" command with the copied link from previous step to download the zip file on OpenDaylight server.

curl -XGET -O
https://nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/integration/karaf/0.16.3/karaf-0.16.3.zip

```
odl-admin@odl-server:~$ curl -XGET -O https://nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/integration/karaf/0.16.3/karaf
-0.16.3.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  221M  100  221M    0     0  9451k      0  0:00:23  0:00:23 --:--:-- 11.8M
odl-admin@odl-server:~$
```

Unzip the downloaded file with the following command

```
~$ unzip karaf-0.16.3.zip
```

Step4: Run OpenDaylight

After unzipping under the same folder, there should be a new directory with the name "**karaf-0.16.3/**".

```
odl-admin@odl-server:~$ ll
total 226436
drwxr-x--- 5 odl-admin odl-admin      4096 Mar  4 19:46 ./
drwxr-xr-x 3 root      root           4096 Mar  4 18:05 ../
-rw------- 1 odl-admin odl-admin       489 Mar  4 19:46 .bash_history
-rw-r--r-- 1 odl-admin odl-admin       220 Jan  6  2022 .bash_logout
-rw-r--r-- 1 odl-admin odl-admin      3823 Mar  4 20:30 .bashrc
drwx------ 2 odl-admin odl-admin      4096 Mar  4 18:13 .cache/
drwxrwxr-x 9 odl-admin odl-admin      4096 Mar  4 19:46 karaf-0.16.3/
-rw-rw-r-- 1 odl-admin odl-admin 231833138 Mar  4 19:33 karaf-0.16.3.zip
-rw-r--r-- 1 odl-admin odl-admin       807 Jan  6  2022 .profile
drwx------ 2 odl-admin odl-admin      4096 Mar  4 20:43 .ssh/
-rw-r--r-- 1 odl-admin odl-admin         0 Mar  4 18:15 .sudo_as_admin_successful
odl-admin@odl-server:~$
```

Assuming download and unzip processes have happened under the home directory, then the path to executable OpenDaylight file is:

```
~$ ~/karaf-0.16.3/bin/karaf
```

When we run it, we will see an output like the following picture which indicate OpenDaylight is running and was installed successfully.


*Figure 6.3 OpenDaylight successful execution*

Any command we enter from now on will be in under OpenDaylight Console. To exit OpenDaylight and get back to shell we can use '**system:shutdown**' or '**logout**'.

# 7. Appendix B: Long outputs

## 7.1 Arp table entry output


```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:Junos="http://xml.juniper.net/Junos/20.1R0/Junos">
<arp-table-information xmlns="http://xml.juniper.net/Junos/20.1R0/Junos-arp"
Junos:style="normal">
<arp-table-entry>
<mac-address>
00:0c:29:ae:55:2e
</mac-address>
<ip-address>
128.0.0.16
</ip-address>
<hostname>
fpc0
</hostname>
<interface-name>
em1.0
</interface-name>
<arp-table-entry-flags>
<none/>
</arp-table-entry-flags>
</arp-table-entry>
<arp-table-entry>
<mac-address>
00:0c:29:d0:15:98
</mac-address>
<ip-address>
192.168.1.1
</ip-address>
<hostname>
192.168.1.1
</hostname>
<interface-name>
fxp0.0
</interface-name>
<arp-table-entry-flags>
<none/>
</arp-table-entry-flags>
</arp-table-entry>
<arp-table-entry>
<mac-address>
00:0c:29:8f:b8:4a
```

*</mac-address>*
*<ip-address>*
*192.168.1.3*
*</ip-address>*
*<hostname>*
*192.168.1.3*
*</hostname>*
*<interface-name>*
*fxp0.0*
*</interface-name>*
*<arp-table-entry-flags>*
*<none/>*
*</arp-table-entry-flags>*
*</arp-table-entry>*
*<arp-table-entry>*
*<mac-address>*
*00:0c:29:f8:66:07*
*</mac-address>*
*<ip-address>*
*192.168.1.4*
*</ip-address>*
*<hostname>*
*192.168.1.4*
*</hostname>*
*<interface-name>*
*fxp0.0*
*</interface-name>*
*<arp-table-entry-flags>*
*<none/>*
*</arp-table-entry-flags>*
*</arp-table-entry>*
*<arp-table-entry>*
*<mac-address>*
*00:0c:29:2e:2f:e7*
*</mac-address>*
*<ip-address>*
*192.168.1.5*
*</ip-address>*
*<hostname>*
*192.168.1.5*
*</hostname>*
*<interface-name>*
*fxp0.0*
*</interface-name>*
*<arp-table-entry-flags>*
*<none/>*
*</arp-table-entry-flags>*
*</arp-table-entry>*

***<arp-entry-count>***
***5***
***</arp-entry-count>***
***</arp-table-information>***
***</nc:rpc-reply>***
***]]>]]>***

## 7.2 Full NETCONF capabilities advertise by NETCONF device


```
{
  "topology": [
    {
      "topology-id": "topology-netconf",
      "node": [
        {
          "node-id": "junipervmx1",
          "netconf-node-topology:connection-status": "connected",
          "netconf-node-topology:port": 830,
          "netconf-node-topology:available-capabilities": {
            "available-capability": [
              {
                "capability": "urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,file",
                "capability-origin": "device-advertised"
              },
              {
                "capability": "urn:ietf:params:netconf:capability:confirmed-commit:1.0",
                "capability-origin": "device-advertised"
              },
              {
                "capability": "urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:1.0",
                "capability-origin": "device-advertised"
              },
              {
                "capability": "urn:ietf:params:xml:ns:netconf:capability:validate:1.0",
                "capability-origin": "device-advertised"
              },
              {
                "capability": "http://xml.juniper.net/netconf/Junos/1.0",
                "capability-origin": "device-advertised"
              },
              {
                "capability": "urn:ietf:params:netconf:base:1.0",
                "capability-origin": "device-advertised"
              },
              {
                "capability": "urn:ietf:params:netconf:capability:candidate:1.0",
                "capability-origin": "device-advertised"
              },
              {
                "capability": "urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring",
                "capability-origin": "device-advertised"
              },
              {
                "capability": "http://xml.juniper.net/dmi/system/1.0",
```

```
                "capability-origin": "device-advertised"
            },
            {
                "capability": "urn:ietf:params:xml:ns:netconf:capability:url:1.0?scheme=http,ftp,file",
                "capability-origin": "device-advertised"
            },
            {
                "capability": "urn:ietf:params:netconf:capability:validate:1.0",
                "capability-origin": "device-advertised"
            },
            {
                "capability": "urn:ietf:params:xml:ns:netconf:capability:candidate:1.0",
                "capability-origin": "device-advertised"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/dhcpv6?revision=2019-01-01)Junos-
rpc-dhcpv6"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/monitor?revision=2019-01-01)Junos-
rpc-monitor"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/l2cpd?revision=2019-01-01)Junos-rpc-
l2cpd"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/jdaf?revision=2019-01-01)Junos-rpc-
jdaf"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/rip?revision=2019-01-01)Junos-rpc-rip"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/amt?revision=2019-01-01)Junos-rpc-
amt"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/security?revision=2019-01-01)Junos-
rpc-security"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/ted?revision=2019-01-01)Junos-rpc-
ted"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/class-of-service?revision=2019-01-
01)Junos-conf-class-of-service"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/iccp?revision=2019-01-01)Junos-rpc-
iccp"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/igmp?revision=2019-01-01)Junos-rpc-
igmp"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/extensible-subscriber-
services?revision=2019-01-01)Junos-rpc-extensible-subscriber-services"
            },
```

```
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/dhcp?revision=2019-01-01)Junos-rpc-
dhcp"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/redundant-trunk-group?revision=2019-
01-01)Junos-rpc-redundant-trunk-group"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/vpls?revision=2019-01-01)Junos-rpc-
vpls"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/mac-rewrite?revision=2019-01-
01)Junos-rpc-mac-rewrite"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/pim?revision=2019-01-01)Junos-rpc-
pim"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/event-viewer?revision=2019-01-
01)Junos-rpc-event-viewer"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/restart?revision=2019-01-01)Junos-rpc-
restart"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/vmhost?revision=2019-01-01)Junos-
rpc-vmhost"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/ppp?revision=2019-01-01)Junos-rpc-
ppp"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/syslog-filenames?revision=2019-01-
01)Junos-rpc-syslog-filenames"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/file?revision=2019-01-01)Junos-rpc-file"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/mac-refresh?revision=2019-01-
01)Junos-rpc-mac-refresh"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/op?revision=2019-01-01)Junos-rpc-op"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/l2-learning?revision=2019-01-01)Junos-
rpc-l2-learning"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/rollback-config?revision=2019-01-
01)Junos-rpc-rollback-config"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/conf/forwarding-options?revision=2019-01-
01)Junos-conf-forwarding-options"
```

```
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/save?revision=2019-01-01)Junos-rpc-
save"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/captive-portal?revision=2019-01-
01)Junos-rpc-captive-portal"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/ping?revision=2019-01-01)Junos-rpc-
ping"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/arp?revision=2019-01-01)Junos-rpc-
arp"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/route?revision=2019-01-01)Junos-rpc-
route"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/services?revision=2019-01-01)Junos-
rpc-services"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/cli?revision=2019-01-01)Junos-rpc-cli"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/conf/unified-edge?revision=2019-01-
01)Junos-conf-unified-edge"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/conf/applications?revision=2019-01-
01)Junos-conf-applications"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/host?revision=2019-01-01)Junos-rpc-
host"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/policer?revision=2019-01-01)Junos-rpc-
policer"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/event-options?revision=2019-01-
01)Junos-rpc-event-options"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/loop-detect?revision=2019-01-
01)Junos-rpc-loop-detect"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/synchronous-ethernet?revision=2019-
01-01)Junos-rpc-synchronous-ethernet"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/multi-chassis?revision=2019-01-
01)Junos-rpc-multi-chassis"
                },
                {
```

```
                "capability": "(http://yang.juniper.net/Junos/rpc/ntp?revision=2019-01-01)Junos-rpc-
ntp"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/system?revision=2019-01-01)Junos-
conf-system"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/agent?revision=2019-01-01)Junos-rpc-
agent"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/root?revision=2019-01-01)Junos-conf-
root"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/ephemeral-
configuration?revision=2019-01-01)Junos-rpc-ephemeral-configuration"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/l2circuit?revision=2019-01-01)Junos-
rpc-l2circuit"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/access-profile?revision=2019-01-
01)Junos-conf-access-profile"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/vlans?revision=2019-01-01)Junos-conf-
vlans"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/firewall?revision=2019-01-01)Junos-
conf-firewall"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/mpls?revision=2019-01-01)Junos-rpc-
mpls"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/ripng?revision=2019-01-01)Junos-rpc-
ripng"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/chassis?revision=2019-01-01)Junos-rpc-
chassis"
            },
            {
                "capability": "(urn:ietf:params:xml:ns:netconf:base:1.0?revision=2011-06-01)ietf-netconf",
                "capability-origin": "device-advertised"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/nonstop-routing?revision=2019-01-
01)Junos-rpc-nonstop-routing"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/mvpn?revision=2019-01-01)Junos-rpc-
mvpn"
            },
            {
```

```
                "capability": "(http://yang.juniper.net/Junos/rpc/ethernet-switching?revision=2019-01-01)Junos-rpc-ethernet-switching"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/ilmi?revision=2019-01-01)Junos-rpc-ilmi"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/load?revision=2019-01-01)Junos-rpc-load"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/routing-instances?revision=2019-01-01)Junos-conf-routing-instances"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/authentication-whitelist?revision=2019-01-01)Junos-rpc-authentication-whitelist"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/firewall?revision=2019-01-01)Junos-rpc-firewall"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/logical-systems?revision=2019-01-01)Junos-conf-logical-systems"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/routing-options?revision=2019-01-01)Junos-conf-routing-options"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/analytics?revision=2019-01-01)Junos-rpc-analytics"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/dvmrp?revision=2019-01-01)Junos-rpc-dvmrp"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/bgp?revision=2019-01-01)Junos-rpc-bgp"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/mvrp?revision=2019-01-01)Junos-rpc-mvrp"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/ospf?revision=2019-01-01)Junos-rpc-ospf"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/task?revision=2019-01-01)Junos-rpc-task"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/multi-chassis?revision=2019-01-01)Junos-conf-multi-chassis"
            },
            {
```

```
                "capability": "(http://yang.juniper.net/Junos/rpc/helper?revision=2019-01-01)Junos-rpc-
helper"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/bridge?revision=2019-01-01)Junos-rpc-
bridge"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/ssl-certificates?revision=2019-01-
01)Junos-rpc-ssl-certificates"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/dot1x?revision=2019-01-01)Junos-rpc-
dot1x"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/policy-options?revision=2019-01-
01)Junos-conf-policy-options"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/sap?revision=2019-01-01)Junos-rpc-
sap"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/access?revision=2019-01-01)Junos-
conf-access"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/test?revision=2019-01-01)Junos-rpc-
test"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/ancp?revision=2019-01-01)Junos-rpc-
ancp"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/auto-bandwidth?revision=2019-01-
01)Junos-rpc-auto-bandwidth"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/spanning-tree?revision=2019-01-
01)Junos-rpc-spanning-tree"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/static-subscribers?revision=2019-01-
01)Junos-rpc-static-subscribers"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/vmhost?revision=2019-01-01)Junos-
conf-vmhost"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/dynamic-tunnels?revision=2019-01-
01)Junos-rpc-dynamic-tunnels"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/services?revision=2019-01-01)Junos-
conf-services"
            },
            {
```

```
                "capability": "(http://yang.juniper.net/Junos/rpc/access-security?revision=2019-01-01)Junos-rpc-access-security"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/accounting?revision=2019-01-01)Junos-rpc-accounting"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/help?revision=2019-01-01)Junos-rpc-help"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/extension-provider?revision=2019-01-01)Junos-rpc-extension-provider"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/class-of-service?revision=2019-01-01)Junos-rpc-class-of-service"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/extension-service?revision=2019-01-01)Junos-rpc-extension-service"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/l2vpn?revision=2019-01-01)Junos-rpc-l2vpn"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/passive-monitoring?revision=2019-01-01)Junos-rpc-passive-monitoring"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/log?revision=2019-01-01)Junos-rpc-log"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/conf/event-options?revision=2019-01-01)Junos-conf-event-options"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/conf/bridge-domains?revision=2019-01-01)Junos-conf-bridge-domains"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/conf/diameter?revision=2019-01-01)Junos-conf-diameter"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/forwarding-options?revision=2019-01-01)Junos-rpc-forwarding-options"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/system?revision=2019-01-01)Junos-rpc-system"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/virtual-chassis?revision=2019-01-01)Junos-rpc-virtual-chassis"
              },
              {
                "capability": "(http://yang.juniper.net/Junos/rpc/app-engine?revision=2019-01-01)Junos-rpc-app-engine"
```

```
        },
        {
          "capability": "(http://yang.juniper.net/Junos/rpc/connections?revision=2019-01-
01)Junos-rpc-connections"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/rpc/vlans?revision=2019-01-01)Junos-rpc-
vlans"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/rpc/aps?revision=2019-01-01)Junos-rpc-
aps"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/conf/multicast-snooping-
options?revision=2019-01-01)Junos-conf-multicast-snooping-options"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/rpc/poe?revision=2019-01-01)Junos-rpc-
poe"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/rpc/hfrr?revision=2019-01-01)Junos-rpc-
hfrr"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/common/types?revision=2019-01-01)Junos-
common-types"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/rpc/snmp?revision=2019-01-01)Junos-rpc-
snmp"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/rpc/lacp?revision=2019-01-01)Junos-rpc-
lacp"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/rpc/rsvp?revision=2019-01-01)Junos-rpc-
rsvp"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/rpc/bfd?revision=2019-01-01)Junos-rpc-
bfd"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/rpc/sflow?revision=2019-01-01)Junos-rpc-
sflow"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/rpc/ospf3?revision=2019-01-01)Junos-rpc-
ospf3"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/conf/chassis?revision=2019-01-01)Junos-
conf-chassis"
        },
        {
          "capability": "(http://yang.juniper.net/Junos/rpc/shmlog?revision=2019-01-01)Junos-
rpc-shmlog"
        },
```

```
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/protection-group?revision=2019-01-
01)Junos-rpc-protection-group"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/validation?revision=2019-01-01)Junos-
rpc-validation"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/diameter?revision=2019-01-01)Junos-
rpc-diameter"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/evpn?revision=2019-01-01)Junos-rpc-
evpn"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/network-access?revision=2019-01-
01)Junos-rpc-network-access"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/mld?revision=2019-01-01)Junos-rpc-
mld"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/unified-edge?revision=2019-01-
01)Junos-rpc-unified-edge"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/file-mgd?revision=2019-01-01)Junos-
rpc-file-mgd"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/conf/snmp?revision=2019-01-01)Junos-
conf-snmp"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/conf/interfaces?revision=2019-01-01)Junos-
conf-interfaces"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/dynamic-profile?revision=2019-01-
01)Junos-rpc-dynamic-profile"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/conf/protocols?revision=2019-01-01)Junos-
conf-protocols"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/clear?revision=2019-01-01)Junos-rpc-
clear"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/v4ov6-tunnels?revision=2019-01-
01)Junos-rpc-v4ov6-tunnels"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/ovsdb?revision=2019-01-01)Junos-rpc-
ovsdb"
                },
                {
```

```
                "capability": "(http://yang.juniper.net/Junos/rpc/set?revision=2019-01-01)Junos-rpc-set"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/programmable-rpd?revision=2019-01-
01)Junos-rpc-programmable-rpd"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/msdp?revision=2019-01-01)Junos-rpc-
msdp"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/virtual-chassis?revision=2019-01-
01)Junos-conf-virtual-chassis"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/path-computation-client?revision=2019-
01-01)Junos-rpc-path-computation-client"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/pgm?revision=2019-01-01)Junos-rpc-
pgm"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/auto-configuration?revision=2019-01-
01)Junos-rpc-auto-configuration"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/subscribers?revision=2019-01-
01)Junos-rpc-subscribers"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/multicast?revision=2019-01-01)Junos-
rpc-multicast"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/traceroute?revision=2019-01-01)Junos-
rpc-traceroute"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/spring-traffic-
engineering?revision=2019-01-01)Junos-rpc-spring-traffic-engineering"
            },
            {
                "capability": "(urn:ietf:params:xml:ns:yang:ietf-inet-types?revision=2013-07-15)ietf-inet-
types",
                "capability-origin": "device-advertised"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/link-management?revision=2019-01-
01)Junos-rpc-link-management"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/ptp?revision=2019-01-01)Junos-rpc-
ptp"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/ddos-protection?revision=2019-01-
01)Junos-rpc-ddos-protection"
            },
            {
```

```
                "capability": "(http://yang.juniper.net/Junos/rpc/ipv6?revision=2019-01-01)Junos-rpc-
ipv6"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/esis?revision=2019-01-01)Junos-rpc-
esis"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/request?revision=2019-01-01)Junos-
rpc-request"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/database-replication?revision=2019-01-
01)Junos-rpc-database-replication"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/switch-options?revision=2019-01-
01)Junos-conf-switch-options"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/fabric?revision=2019-01-01)Junos-
conf-fabric"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/poe?revision=2019-01-01)Junos-conf-
poe"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/conf/security?revision=2019-01-01)Junos-
conf-security"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/version?revision=2019-01-01)Junos-
rpc-version"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/pppoe?revision=2019-01-01)Junos-rpc-
pppoe"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/access-cac?revision=2019-01-01)Junos-
rpc-access-cac"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/pfe?revision=2019-01-01)Junos-rpc-pfe"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/dhcp-security?revision=2019-01-
01)Junos-rpc-dhcp-security"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/isis?revision=2019-01-01)Junos-rpc-
isis"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/ingress-replication?revision=2019-01-
01)Junos-rpc-ingress-replication"
            },
            {
                "capability": "(http://yang.juniper.net/Junos/rpc/backup-selection?revision=2019-01-
01)Junos-rpc-backup-selection"
```

```
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/lldp?revision=2019-01-01)Junos-rpc-
lldp"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/oam?revision=2019-01-01)Junos-rpc-
oam"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/r2cp?revision=2019-01-01)Junos-rpc-
r2cp"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/vrrp?revision=2019-01-01)Junos-rpc-
vrrp"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/quit?revision=2019-01-01)Junos-rpc-
quit"
                },
                {
                  "capability": "(urn:ietf:params:netconf:base:1.0?revision=2011-06-01)ietf-netconf"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/conf/accounting-options?revision=2019-01-
01)Junos-conf-accounting-options"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/conf/dynamic-profiles?revision=2019-01-
01)Junos-conf-dynamic-profiles"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/performance-
monitoring?revision=2019-01-01)Junos-rpc-performance-monitoring"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/as-path?revision=2019-01-01)Junos-
rpc-as-path"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/interfaces?revision=2019-01-01)Junos-
rpc-interfaces"
                },
                {
                  "capability": "(http://yang.juniper.net/Junos/rpc/ldp?revision=2019-01-01)Junos-rpc-
ldp"
                }
              ]
            },
            "netconf-node-topology:host": "192.168.1.2"
          }
        ]
      }
    ]
}
```

# 8. References


[1] Patricia Morreale, James Anderson, “Software Defined Networking: Design and Deployment”, Publisher: CRC Press, Year: 2014 ISBN: 978-1-4822-3863-1,1482238632,978-1-4822-3864-8

[2] E. Haleplidis, Ed., K. Pentikousis, Ed., S. Denazis, J. Hadi Salim, D. Meyer, O. Koufopavlou, “Software-Defined Networking (SDN): Layers and Architecture Terminology”, Internet Research Task Force (IRTF)RFC7426, JAN 2015, https://www.rfc-editor.org/rfc/rfc7426

[3] [Online]. Available: https://opennetworking.org/sdn-definition/

[4] [Online]. Available: https://www.knowledgenile.com/blogs/software-defined-networking-architecture-pros-and-cons-of-sdn/

[5] [Online]. Available: https://zindagitech.com/what-are-the-advantages-disadvantages-and-architectural-components-of-sdn/

[6] [Online], Available: https://www.lightreading.com/opendaylight-project-founded/d/d-id/702110

[7] [Online], Available: https://thenewstack.io/sdn-series-part-vi-opendaylight/

[8] [Online], Opendaylight Project. Available: http://www.opendaylight.org/

[9] Ahmad Hemid, “Facilitation of The OpenDaylight Architecture”, Bonn-Rhein-Sieg University of Applied Sciences, https://www.researchgate.net/publication/317057083_Facilitation_of_The_OpenDaylight_Architecture

[10] [Online], Available: https://wiki.opendaylight.org/display/ODL/MD-SAL

[11] [Online], Available: https://www.fir3net.com/Networking/Protocols/an-introduction-to-netconf-yang.html

[12] [Online], Available: https://en.wikipedia.org/wiki/YANG

[13] M. Bjorklund, Ed., Tail-f Systems, “The YANG 1.1 Data Modeling Language”, Internet Engineering Task Force (IETF) RFC 7950, Available: https://www.rfc-editor.org/rfc/rfc7950

[14] [Online], Available: http://YANG.ciscolive.com/pod0/labs/lab2/lab2-m1

[15] M. Bjorklund, Ed., Tail-f Systems, “YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)”, Internet Engineering Task Force (IETF) RFC 6020, Available: https://www.rfc-editor.org/rfc/rfc6020.html

[16] [Online],Hakan Millroth, “Instant YANG” by Tail-f, Available: https://www.tail-f.com/wordpress/wp-content/uploads/2014/02/Tail-f-Instant-YANG.pdf

[17] [Online], Available: https://www.tail-f.com/what-is-netconf/

[18] [Online], Available: https://info.support.huawei.com/info-finder/encyclopedia/en/NETCONF.html

[19] R. Enns, Ed., M. Bjorklund, Ed., J. Schoenwaelder, Ed., A. Bierman, Ed., “Network Configuration Protocol (NETCONF)”, Internet Engineering Task Force (IETF) RFC6241, Available: https://www.rfc-editor.org/rfc/rfc6241

[20] M. Wasserman, T. Goddard, “Using the NETCONF Configuration Protocol over Secure Shell (SSH)”, Network working group RFC4742, Available: https://www.rfc-editor.org/rfc/rfc4742.txt

[21] T.Goddard, “Using NETCONF over the Simple Object Access Protocol (SOAP”, Network working group RFC 4743, Available: https://www.rfc-editor.org/rfc/rfc4743

[22] M. Badra, “NETCONF over Transport Layer Security (TLS)”, Network working group RFC5539, Available: https://www.rfc-editor.org/rfc/rfc5539.txt

[23] M. Wasserman, “Using the NETCONF Protocol over Secure Shell (SSH)”, IETF RFC6242, Available: https://www.rfc-editor.org/rfc/rfc6242

[24] [Online], Available: https://www.juniper.net/documentation/us/en/software/junos/automation-scripting/topics/concept/junos-script-automation-junos-os-xml-overview.html

[25] [Online], Available: https://ultraconfig.com.au/blog/how-to-configure-juniper-routers-with-netconf-via-xml-rpcs/

[26] [Online], “Establish an SSH Connection for a NETCONF Session”, Available: https://www.juniper.net/documentation/us/en/software/Junos/netconf/topics/topic-map/netconf-ssh-connection.html

[27] [Online], “Configure Interoperability Between MX Series Routers and OpenDaylight”, Available: https://www.juniper.net/documentation/us/en/software/junos/netconf/topics/task/configure-odl-integration.html

[28] [Online], Available: https://www.juniper.net/documentation/us/en/software/Junos/netconf/topics/task/netconf-configuration-setting-edit-config-mode.html

[29] [Online], Available: https://www.juniper.net/documentation/us/en/software/Junos/netconf/topics/task/netconf-configuration-editing.html

[30] [Online], Available: https://docs.opendaylight.org/projects/netconf/en/latest/user-guide.html