Virtual lab for SDN and NFV - implementation and testing

Juraj Londák, Martin Medvecký, Peter Maduda, Pavol Podhradský [1]
[1] Slovak University of Technology in Bratislava, Ilkovičova 3, 812 19, Bratislava, Slovakia
martin.medvecky@stuba.sk

Abstract - **This paper describes the process and experience of implementing and testing a virtual laboratory. The laboratory enables to install, configure and test selected SDN and NFV networks designed for both research and education.**

Keywords – **SDN, NFV**

I. INTRODUCTION

SDN (Software Defined Networking) is a network design concept with the physical separation of the network management plane from the transport layer. SDN is probably one of the most significant evolutionary trends in ICT. Developments in SDN and NFV (Network Funcion Virtualisation) are related to new network requirements, evolving trends in remote management of digital data, applications and IT services.

SDN is programmable and centrally controlled, i.e. network intelligence is centralized in a software controller to provide global oversight over the entire network that appears to applications as a logic switch. SDN concept is based on the following three components:

- SDN applications - programs that communicate via an API with an SDN controller. In addition, applications can collect information from the controller for decision-making purposes.
- SDN controller - a logical unit that receives instructions from the application layer and transfers it to network elements. This communication also works in the opposite direction, i.e., the controller extracts data from hardware network devices and transfers them back to applications.
- SDN network devices - provide network data transmission.

SDN is based on open standards and devices from different vendors - new implementations are implemented through open standards, simplifying the construction and performance of the network because commands are provided by the SDN controller instead of multiple specific devices from different vendors.

The architecture of NFV technology was designed by Network Functions Virtualization Industry Specification Group (ETSI NFV ISG) based on the following components:

- NFVI (NFV Infrastructure) - provides virtual resources needed to support the implementation of virtualized network functions,
- VNF (Virtualized Network Functions) - software implementation of network functions that is able to run by NFVI,
- NFV MANO (Management and Orchestration) - covers orchestration and lifecycle management of physical and / or software tools that support the virtualization and infrastructure lifecycle management.

II. VIRTUAL LAB FOR SDN AND NFV - DESCRIPTION

The system architecture of the Virtual Laboratory presented in [5] is depicted on Fig. 1. This design is characterized by an emphasis on high availability and includes device redundancy. The proposed system architecture supports the implementation of OpenStack, OpenDaylight and OPNFV platforms.

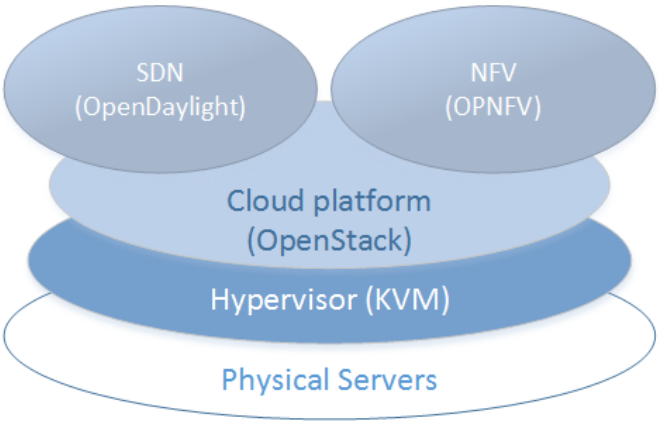

Figure 1. General architecture of SDN and NFV virtual lab

OpenStack is a cloud-based software platform that provides simple implementation, high-availability and support for a large number of features. It does not have any proprietary hardware or software requirements and is designed to work with both fully-virtual and bare-metal systems.

OpenDaylight is a modular open source SDN platform developed to promote SDN and NFV. OpenDaylight enables network services across a spectrum of hardware in multivendor environments.

OPNFV (Open Platform for NFV) brings together upstream components across compute, storage and network virtualization in order create an end-to-end platform. The present OPNFV

release, Danube, builds and integrates multiple end-to-end networking stacks, including MANO, data plane acceleration, and architecture advancements.

III. IMPLENTATION OF VIRTUAL LAB FOR SDN AND NFV

For the purpose of the project, several different HP servers and several network elements were allocated. The first task was to design the deployment of OpenStack entities on available servers, and then to configure and integrate servers and network elements and to place them in a rack.

The largest number of services will be run on the control node, and therefore the highest performance requirements will be placed on this server. Therefore, the most powerful available server (HP ProLiant DL360 G8) server was selected as the controller. In the future, it is planned to install a second controller, which will allow the creation of an identical control node, which will increase the reliability and availability of OpenStack.

The storage node was implemented on available network storage HP StorageWorks X1600 equipped with sufficient disk capacity (9 pieces of 1TB SATA drives). The Cinder is installed on the network storage node to implement the block store function.

Other servers - the HP ProLiant DL380 G5 and the HP ProLiant DL380 G5, even if they are not as powerful as the control node server, serve as computing nodes with Nova service.

The architecture thus proposed will allow future addition of such computational nodes to the environment and increase the computational resources of the entire environment. The big advantage of all servers is a support of hardware virtualization, so it is possible to use a KVM hypervisor that uses Kernel Linux as a hypervisor (more powerful than other standalone hypervisors on servers without hardware virtualization).

Figure 2. Rack with virtual lab hardware

The router Mikrotik RB 1100 AH will be used to connect to the external network and we will use two switches ASUS GigaX2024 on the internal network.

All servers and network elements were placed in a rack. The physical location of the servers and the design of the OpenStack services that are placed on them will be seen in the Fig.2.

A. *OpenStack Implementation*

In the first step of the implementation phase we focused our research activities on the OpenStack platform implementation. As it is illustrated in Fig. 2, OpenStack contains multiple components. It therefore has a wide range of applicability from cloud service providers, to large institutions that use it to create private clouds.

Synchronization of services

For proper synchronization of individual OpenStack services, it is necessary to ensure accurate, uniform set-up of the internal clock for all nodes. We will secure this sync using the NTP protocol. The implementation of NTP Chrony was used. Chrony is a simple and versatile NTP implementation that allows to synchronize internal system clocks with designated NTP servers, external reference clocks (e.g. GPS), or manually enter time using the keyboard.

In OpenStack, it is necessary to have exact synchronization between its nodes, in case of inaccuracies in time between the nodes there may be error states, the message broker is especially vulnerable to this. Therefore, with such implementation, much greater emphasis is placed on synchronizing nodes in the internal environment than on synchronization with the outside world. An effective way to achieve such internal synchronization is to place another NTP server directly into this environment (most often directly on the control node) through which other nodes will be synchronized. In our laboratory, we have an available NTP server directly on the Mikrotik RB1100 router (Layer 4). According to this server, we will synchronize the internal NTP chrony server created on the control node (controller), which will also synchronize the other OpenStack nodes (Fig. 3).

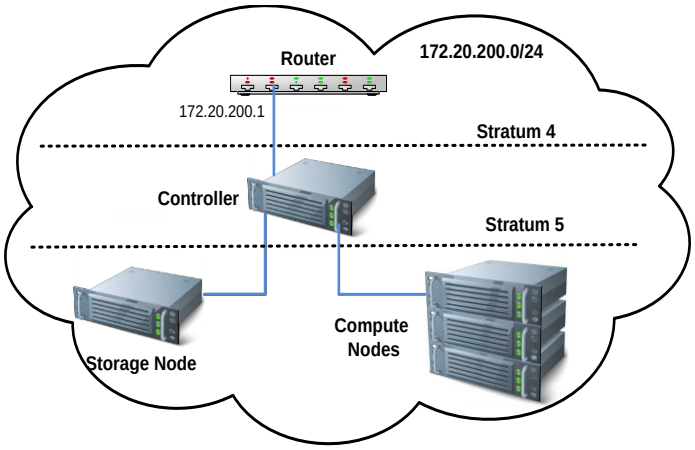

Figure 3. NTP hierarchy

SQL database

A lot of OpenStack services use a SQL database to store information. The MariaDB database is used in the project. Before installing OpenStack itself, it is necessary to install and prepare the SQL database. The database itself will run on *controller*, as it will run most of the services that use it. First of all, we need to install the mariaDB and python SQL client required for OpenStack:

user@controller:~$apt install mariadb-server python-pymysql

It was then necessary to properly perform the initial configuration of the database. Configuration can be done using a configuration file *99-openstack.cnf* located at */etc/mysql/mariadb.conf.d.*

Message queues

OpenStack uses message queues to coordinate operations and distribute state information between services. Individual services do not communicate directly with each other, but they communicate using AQMP (Advanced Message Queuing Protocol) through so called message brokers. AQMP is an application layer protocol using TCP for reliable messaging. Transmission can be secured using TLS (SSL). The main purpose of the AMQP protocol is to modify and manage the publisher and subscriber's reports to the extent that communication is universal and multi-vendor products can be used. The AMQP model has the following general view: messages are delivered to message exchange entities, which are often compared to post offices or mailboxes. Message Exchange Entities then distribute the copies of the stacks into queues using certain rules (rules are called "bindings"). AMQP brokers then deliver a message to customers subscribed to queues or their subscribers download them on request (fetch/pull). This model is shown in Fig. 4.

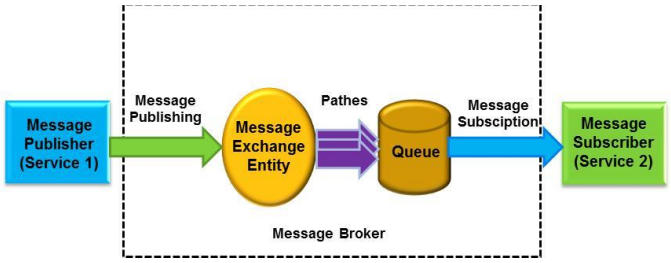

Figure 4. AQMP protocol model

A RabbitMQ has been used as a AMQP broker in virtual lab, because it is the open source message broker software supported in the largest number of distributions and therefore has a great support for the OpenStack community. The RabbitMQ broker has been installed and connected to *controller*.

Memcached

Memcached is a generic distributed memory data caching system. It is used to accelerate dynamically managed web sites by storing data and objects in RAM, reducing the number of external data reads. Memcached is a memory cache daemon that can be used by most OpenStack services to store temporary data i.e. tokens. Authentication mechanism of the identity services for other OpenStack services (Keystone) uses Memcached to store tokens in memory cache. Memcached has been installed as similar supplementary and support elements to the control node *controller*.

OpenStack packages

Different Linux distributions publish OpenStack packages directly as part of the distribution, or separately in case of disagreement on release dates. All nodes in the virtual lab have a Ubuntu server 16.04.2 LTS installed. Therefore, before installing and configuring individual OpenStack services on nodes, it was necessary to install OpenStack packages on all nodes.

Identity service - Keystone

Keystone is an OpenStack service that provides API client authentication, service discovery, and distributed multi-tenant authorization by implementing OpenStack's Identity API.

The HTTP Apache server with mod_wsgi was used to configure the Keystone service, because if the Keystone package is used that the entire configuration of the Apache server and the mod_wsgi module activation is directly controlled by the Keystone package. The Apache server handles requests for the identity service on ports 5000 and 35357 (which are also the ports on which Keystone itself is listening). Before configuring the service itself, it was necessary to create a database for the service to store the data. After the database was created, it was necessary to set access to the database both from the node on which it is located and for all other nodes. Each node will be authenticated against the database by a password.

B. *OpenDaylight Implementation*

In the second step of the implementation phase we implemented the OpenDaylight SDN controller. After the OpenDaylight installation (see Fig. 5) selected features have been installed, e.g. by the command *feature:install odl-dlux-all* an intuitive graphical user interface for OpenDaylight was installed.

Currently there are 46 regular and 18 experimental features available. Except DLUX, several other features, e.g. L2 Switch (provides L2 /Ethernet/ forwarding across connected OpenFlow switches) or OVSDB OpenStack Neutron (provides OpenStack Network Virtualization using OpenDaylight's OVSDB support) have been installed.

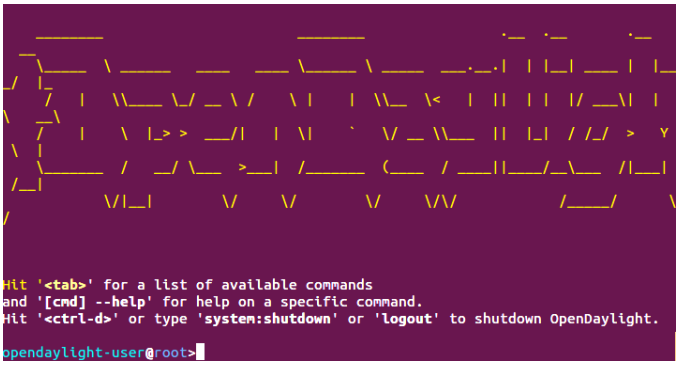

Figure 5. OpenDaylight CLI console

Virtual Lab Network Architecture

Network architecture of Virtual Lab contains multiple separated VLANs, as it is illustrated in Fig. 6. This network separation provides better bandwidth management options and minimizes cross dependencies on each HW nodes during bandwidth consumable operations.

- Management VLAN – serves for the internal communication of OpenStack components and access to the Internet. It also provides Out-of-Band management access to all servers.
- Storage VLAN – Separates communication between a Storage server with block disks and individual instances running on Compute nodes from other network traffic.
- Instance VLAN - serving on inter-instance communication.

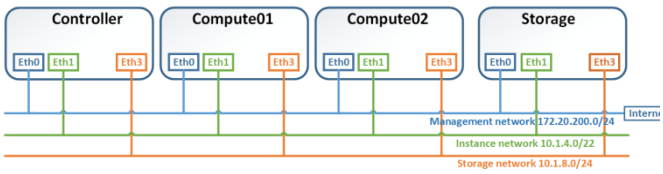

Figure 6. Network architecture of SDN and NFV Virtual laboratory

IV. Testing of Virtual Lab for SDN and NFV

The last step in virtual laboratory implementation consisted of testing the components of OpenStack and OpenDaylight. Each functionality has been tested after it has been deployed, with a specific set of commands to confirm its functionality and interoperability with other related features.

Before testing the OpenStack deployment, a firmware upgrade of all servers was made. Upgrading was done both for safety reasons and in terms of functionality.

The functionality of the OpenDaylight Controller installation was tested via the Mininet network simulator, which was then installed in a separate VM. Testing the ODL controller's functionality was due to its ability to manage a simulated network created by the Mininet simulator.

V. Conclusions

In this paper, the process of the implementation and testing of SDN and NFV virtual laboratory (OpenStack and OpenDaylight) is presented. The future work will be focused on the implementation and testing of the OPNFV

Acknowledgment

Research described in the paper was financially supported by the H2020 project NEWTON, No. 688503 and VEGA project INOMET, No. 1/0800/16

References

[1] OpenFlow and Software Defined Networks Presentation. http://www.openflow.org/documents/OpenFlow_2011.pps

[2] Chiosi, M., Wright, S., Clarke, D., Willis, P., at al.: Network Functions Virtualisation: Network Operator Perspectives on Industry Progress, http://portal.etsi.org/NFV/NFV_White_Paper2.pdf

[3] ETSI GS NFV-SWA 001 V1.1.1: Network Functions Virtualisation: Virtual Network Functions Architecture. France: ETSI, December 2014

[4] ETSI GS NFV-INF 001 V1.1.1: Network Functions Virtualisation: Infrastructure Overview. France: ETSI, January 2015

[5] Londák, J., Medvecký, M., Tóth, T., Podhradský, P.: Virtual lab for SDN and NFV - concept and architecture, in Proceedings of International Workshop Redžur 2017, Bratislava, Slovakia