Implementation and Deployment of a Server at the Edge Using OpenStack Components

Carlo Vitucci
Technology Management
Ericsson AB
Stockholm, Sweden
e-mail: carlo.vitucci@ericsson.com

Tommaso Cucinotta, Riccardo Mancini, Luca Abeni
Real-Time System Laboratory (RETIS)
Scuola Superiore Sant'Anna
Pisa, Italy
e-mail: firstname.lastname@santannapisa.it

***Abstract*—As the 5th telecommunication Generation (5G) deployments are spreading around via various mobile operators, the capabilities behind 5G are becoming more and more understandable. Infrastructure vendors, operators, and end users now have a clear picture of the 5G potential and, for that reason, the research and the development of 5G are surely continuing. The one-to-one mapping between 5G and Software Defined Network - Network Function Virtualization (SDN-NFV) architecture is not in discussion, but the impact of porting SDN-NFV into the Radio Access Network (RAN) is still under investigation. Sometimes, the RAN requirements set strong limitations even in the basic hardware and software setup. For example, the most complete and very well integrated SDN-NFV infrastructure distributions require specific hardware capabilities in terms of available nodes, in contrast with the RAN requirement to be economic, power consumption limited and with limited overhead due to operating system and middleware cost. For that reason, this study uses only a minimal set of OpenStack components in order to evaluate what is the minimal hardware capability needed to set up a basic, but fully working environment for NFV, highlighting the pros and cons of embracing a solution solely based on standard OpenStack components.**

Keywords-5G; RAN; SDN-NFV, edge computing; server at the edge; Service deployment; OpenStack, E2E deployment.

I. INTRODUCTION

2019 is the year in which 5G started to be a practical and viable commercial solution available to mobile operators [1]. The importance of 5G architecture is now widely understood and shared: the new technology has the potential to drive economic growth. Its possibilities are so broad that we probably cannot even imagine what and how many new services will be possible. Today, all operators see 5G as the enabler for full connectivity between people, for the creation of the Internet of Things (IoT) and as a startup for the so-called Industry 4.0. However, although this is already very stimulating and large enough to justify the investment in the new architecture and infrastructure, 5G is beyond all of that. Smart cities, Industrial IoT, augmented reality, autonomous transport, digital health, are just some of the countless commercial opportunities that could be possible when 5G will be fully deployed. To allow such an enormous commercial opportunity to become real, it is necessary to be able to count on a very well-defined ecosystem, where a new approach to the network is needed, including (RAN), to address the wide distribution of functions, applications, and data. The distribution of services requires an End-to-End architecture (E2E) where, thanks to a high-level programmability and "software-ability" of the architecture, it will be possible to offer new advanced services to consumers by dedicating portions of the network. In this mode, it will be possible to guarantee precise levels of service quality and to respond to the increasingly demanding application needs of a wide variety of sectors (Figure 1).

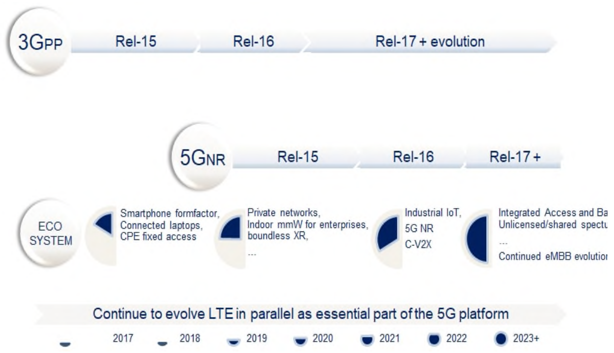

3GPP
Rel-15
Rel-16
Rel-17 + evolution
5GNR
Rel-15
Rel-16
Rel-17 +
ECO SYSTEM
Smartphone formfactor, Connected laptops, CPE fixed access
Private networks, Indoor mmW for enterprises, boundless XR, ...
Industrial IoT, 5G NR C-V2X
Integrated Access and Ba Unlicensed/shared spectu ... Continued eMBB evolutio
Continue to evolve LTE in parallel as essential part of the 5G platform
2017
2018
2019
2020
2021
2022
2023+

Figure 1. 5G Standards and Commercialization Time Line [2]

SDN-NFV is the most suitable system architecture to support the necessary 5G ecosystem [3]. To make it successful, however, the solution must rely on an NFV infrastructure (NFVI) optimized to support the rapid implementation of new generation services with low latency and a varied and distributed group of terminals and devices. As mentioned in our previous work [4], the infrastructure shall be designed to remove inefficiencies in the modern cloud due to a distance between the design of high-level cloud management/orchestration and low-level kernel/hypervisor mechanisms. The two worlds should talk to each other, providing richer abstractions to describe the low-level mechanisms and to automatically map higher-level descriptions and abstractions to configuration and performance tuning options available within operating systems and kernels (both host and guest), as well as hypervisors.

The rest of this paper is organized as follows. Section II introduces the Network Operating System definition and explains the decision to use OpenStack components. Section III describes the hardware environment used in the implementation phase. Section IV addresses the software environment setup. Section V goes into finer details for OpenStack components selection. Section VI and Section VII emphases the set of for network and storage respectively, while Section VIII lines out the deployment configuration actions needed. Eventually, Section IX shows the deployment sequence. Section X points to the hardware minimal capability as those used by the experiment and Section XI discusses some conclusions.

II. Network OS

The Network Operating System (NOS) is, by definition, the horizontal server network resources controller in a distributed system. It is responsible to provide a virtualized (programmable) environment and the connected control part. Describing the structure of the SDN-NFV architecture is not one of the purposes of this paper, but related references are easily available [5][6]. In the SDN-NFV architecture, the NOS is spread between NFVI and Virtual Infrastructure Manager (VIM), as graphically shown in Figure 2.

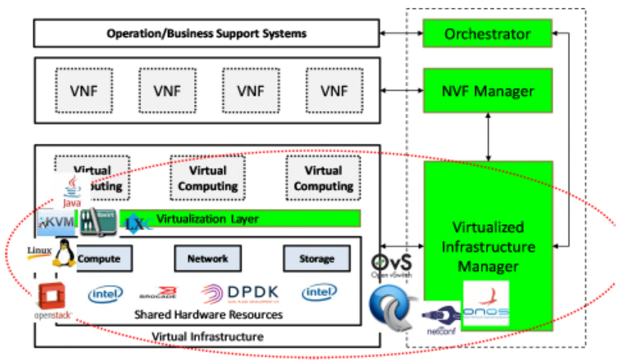


Figure 2. Graphical location of the NOS: network resources control and allocation location in the SDN-NFV architecture are spread between NFVI and VIM

"De facto", commercial solutions use OpenStack to deploy virtualized environments. OpenStack is an "always evolving" project built over several components. These can easily be added or removed from the configuration of a deployment, optionally plugging other open-source components/agents, like OpenDayLight, NetConf, OpenFlow and others [7]. For any next-generation mobile system, a mandatory requirement is to be a fully integrated ecosystem that, independently of specific vendors, can be orchestrated by a (logically) centralized controller. Assuming for the RAN the server configuration described in [3], in the following we describe an OpenStack deployment over a few different boards constituting a simple edge-computing test-bed.

III. Hardware Environment Setup

Hardware environment set up has been done considering some main rules:

1. It shall be, as much as possible, based on commercial hardware and have limited cost;
2. It shall consist of a basic set of hardware components and boards;
3. It shall be suitable for housing a NOS fully based on OpenStack components;

The first rule has been set considering the capillary, widely distributed, explosion of computer deployment close to the end user, into the edge of the network [8][9]: minimizing the cost of the deployment looks like a strong requirement for the success of the 5G implementation.

The second rule has been set to overcome the limits the most popular OpenStack distributions have. For example, Open Platform for NFV (OPNFV), a complete solution for development and evolution of NFV components across various open-source ecosystems, requires a significant number of controller and specific hardware characteristics for the system development board [10]. OPNFV is surely a complete and powerful solution, but, in this work, we are interested in understanding the bare minimum set of hardware characteristics necessary for implementing a NOS based on open software.

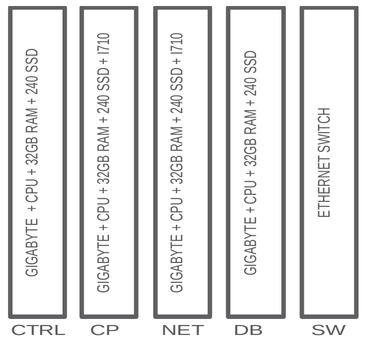

GIGABYTE + CPU + 32GB RAM + 240 SSD
GIGABYTE + CPU + 32GB RAM + 240 SSD + I710
GIGABYTE + CPU + 32GB RAM + 240 SSD + I710
GIGABYTE + CPU + 32GB RAM + 240 SSD
ETHERNET SWITCH
CTRL
CP
NET
DB
SW

Figure 3. Hardware Environment Set up

The third rule is a practical decision (software availability) and it is not limiting the result achieved in the lab. With the only exception of the radio interface board, for which it is possible to use 5G-ready existing radio product solutions, the server at the edge has been built (see Figure 3) as one compute node (CP), one networking node (NET), one controller node (CTRL) and an Ethernet switch (SW). The development & deployment environment is represented by another board, the developer node, that will act also as containers repository site (DB).

TABLE I. HARDWARE COMPONENTS FOR THE NODE

Object	*Vendor & Type*
Motherboard	GIGABYTE H310M-A
CPU	Intel Core i3 8100
Disk	WD Green3D Nand 240 GB
RAM	DDR4 Corsair Vengeance 32 GB
Eth. daughterboard	Intel X710-DA2
Power	SFX Power 2
Router	Netgear ProFase GS108

The hardware characteristics of the nodes are summarized in Table I.

IV. SOFTWARE ENVIRONMENT SETUP

The software environment set up has been done considering only open-source components imposing minimal requirements on the needed underlying hardware. The software shall be able to run into the minimal set of boards used for the server at the edge concept and it shall be fully based on open-source packaging. For the servers at the edge, we chose to use a Linux operating system, Ubuntu [11] distribution. The Deployment Board uses a Desktop version while the other boards use a Server distribution (see Figure 4).

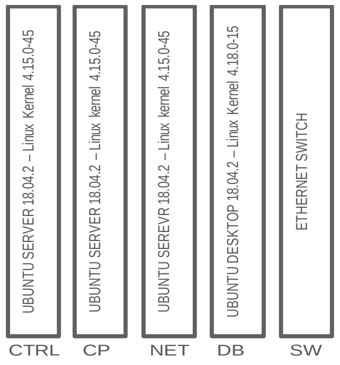

UBUNTU SERVER 18.04.2 – Linux Kernel 4.15.0-45
UBUNTU SERVER 18.04.2 – Linux kernel 4.15.0-45
UBUNTU SEREVR 18.04.2 – Linux kernel 4.15.0-45
UBUNTU DESKTOP 18.04.2 – Linux Kernel 4.18.0-15
ETHERNET SWITCH
CTRL
CP
NET
DB
SW

Figure 4. Hardware Environment Set up

The latest 18.04.2 Ubuntu Long Term Support (LTS) is used. Kernel version is 4.15 for server and 4.18 for Desktop. During the test phase, the node has been regularly upgraded with the Ubuntu standard updates using the apt package manager [12]. At the time when this paper has been written, the latest working update was:

DB: Linux 4.18.0-24-generic #25~18.04.1-Ubuntu SMP

OTHERS: Linux 4.15.0-28-generic #64-Ubuntu SMP

V. INFRASTRUCTURE SETUP

The most complete and up-to-date among available open-source distributions of OpenStack for the infrastructure is probably OPNFV [13], an SDN-NFV distribution fully integrated with the latest technologies, for example, Open Network Automation Platform (ONAP) [14] and Open RAN (O-RAN) [15]. However, the OPNFV hardware requirements are not suitable for an edge-computing proof of concept (PoC), as it requires a minimum of 2 controller nodes, 3 compute nodes, and a minimum of 64 gigabyte (GB) Random-Access Memory (RAM) mounted. For that reason, the PoC building of the server at the edge has been fully based on self-building OpenStack components and we started from the suggested configuration for containers handling [16].

The OpenStack components List is (Figure 5):
Basic Infrastructure components (mandatory)

- Nova, to provide compute instances;
- Glance, to provide an image service;
- Keystone, to provide Application Program Interface (API) client authentication;

Extended Infrastructure components (mandatory)

- Neutron, to provide network connectivity;
- Swift, to provide an objects store service;
- Cinder, to provide block storage and volume service;

Extended infrastructure components (optional)

- Kuryr, network plugin to provide networking services to Docker containers;

Optional enhancements

- Horizon, Dashboard to provide a web-based user interface;
- Grafana, to provide a metrics dashboard;
- Cyborg, to support possibly available accelerations: Graphics Processing Unit (GPU), Data Plane Development Kit (DPDK), etc...

Consumption services

- Tacker, to provide generic VNF Manager (VNFM) and NFV Orchestrator (NFVO);
- Kolla-Ansible, to deploy OpenStack components in Docker containers using Ansible;
- Zun, to provide API for launching and managing containers;
- Magnum, to provide container orchestration services;
- Heat, to provide template-based orchestration.

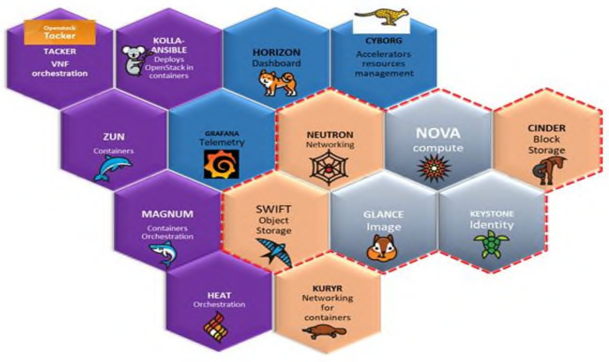


Figure 5. Selected OpenStack Package

Kolla-Ansible [17] has been conveniently used to ease the deployment of the various OpenStack components. It is worth to mention here that the selected set of OpenStack components do not constitute necessarily an optimal selection, as an evaluation or comparison of the possible optional components was not in the goal of the study described in this paper.

Kolla-Ansible comes with a minimal set of software requirements and dependencies on other software components. The list of install dependencies is available in [18].

During the set-up and configuration phase, various issues have been tackled and the following workarounds applied:

- We had to use the development version of Kolla-Ansible because the released version seemed to have issues in the container deployment phase (specifically, raising the MariaDB container didn't work).
- The Internet Protocol (IP) check for services was failing due to the lack of configured passwords. This was fixed by adding password roles in the /etc/sudoers file.
- We needed to install docker-ce instead of docker 12.1.0*. Note that the deployment board is used as a local Docker registry in our environment.
- Create a link -s -L to easy_install in /usr/local/bin/ because it doesn't exist in the installed distribution. You need to compile and install python-3.7 locally.
- During the deployment phase, Koll-Ansible uses frequently Docker commands. This might generate permission denied alarm. In order to remove that issue, it is enough to add own user to the Docker group.

```
sudo gpasswd -a $USER docker
newgrp docker
```

- the local Docker registry address needed to be added to the list of allowed insecure registries in the Docker daemon configuration file (/etc/docker/daemon.json).

The Docker daemon uses the HTTP_PROXY, HTTPS_PROXY and NO_PROXY environment variables. Those variables cannot be configured using the daemon.json file. They can be set in an http-proxy.conf file in the /etc/system/docker.service.d directory. The definition of the NO_PROXY allows contacting the internal Docker register without proxying.

VI. NETWORK SETUP

Kolla-Ansible needs two IP addresses per board: the networking setup for OpenStack is one of the most complicated actions to do, but Kolla-Ansible as a deployment tool is simplifying a lot. We let Kolla-Ansible set neutron for us, with the cost of the setup of two Virtual Local-Area Network (VLAN) per board. Note that our server and desktop distributions are not the same. The server is using netplan while the desktop is still counting on ifupdown. In order to manage the network using the same setting, ifupdown has been installed in our Ubuntu Server nodes. Once ifupdown has been loaded, the vlan configuration could be done by editing the /etc/network/interface file.

```
auto eno1
iface eno1 inet dhcp

auto eno1.1
iface eno1.1 inet static
        address 11.22.33.44/23
        netmask 255.255.254.0
        gateway XX.YY.ZZ.1
        vlan-raw-device eno1

auto eno1.2
iface eno1.2 inet static
        address 11.22.33.55/23
        netmask 255.255.254.0
        gateway 11.22.33.1
        vlan-raw-device eno1

auto lo
iface lo inet loopback
```

Eventually, the vlan kernel module is installed to keep network setting permanent:

```
sudo su -c 'echo "8021q" >> /etc/modules
```

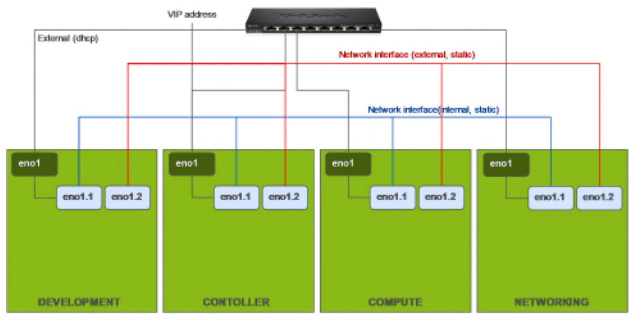
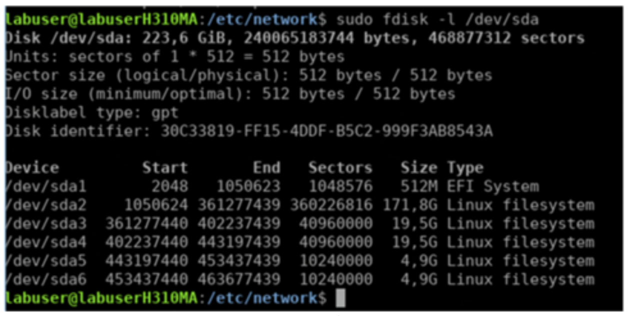


Figure 6. Server at the edge connectivity

The networking is done to have demo-networking, internal, public networking, external and VIP networking, neutron keepalive control, as shown in Figure 6.

VII. STORAGE SETUP

Swift requires block devices to be available for storage. To prepare a disk for using a swift storage device, a special partition name and filesystem label needed to be added. Moreover, before running Swift, we had to generate rings, which are binary compressed files that at a high level let the various Swift services know where data is in the cluster. Cinder also needs a dedicated Logical Volume Management (LVM) physical volume group. Note the partition for swift and Cinder are strongly recommended (see Figure 7).



```
labuser@labuserH310MA:/etc/network$ sudo fdisk -l /dev/sda
Disk /dev/sda: 223,6 GiB, 240065183744 bytes, 468877312 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 30C33819-FF15-4DDF-B5C2-999F3AB8543A

Device          Start       End   Sectors   Size Type
/dev/sda1        2048   1050623   1048576   512M EFI System
/dev/sda2     1050624 361277439 360226816 171,8G Linux filesystem
/dev/sda3   361277440 402237439  40960000  19,5G Linux filesystem
/dev/sda4   402237440 443197439  40960000  19,5G Linux filesystem
/dev/sda5   443197440 453437439  10240000   4,9G Linux filesystem
/dev/sda6   453437440 463677439  10240000   4,9G Linux filesystem
labuser@labuserH310MA:/etc/network$ 
```

Figure 7. The disk partition of the development board

The Swift and Cinder disk partitions are required where the system storage is hosted; as described in the infrastructure setup, this is the development board in this study. Note that the size of the partitions is not optimized, and the correct size should be defined or investigated in advance for the product deployment case. Most likely, a real production environment needs bigger disks/partitions.

VIII. Kolla-Ansible Configuration Files

The most attractive benefit of using the Kolla-Ansible tool to deploy OpenStack is that it provides a very simple procedure to identify and characterize the overall system, both in hardware and software point of view. OpenStack package components set, network setup, hardware inventory and storage definitions are defined and managed using only two files: the so-called "globals.yml" and the "multinode.yml" configuration file. Both of them are available as a template in the Kolla-Ansible distribution/installation file. To match the real hardware setup and use the selected OpenStack components, the customization of them is straightforward: remove or add a comment to existing lines.

"multinode.yml" is the Ansible inventory file and configures the connection parameters for the hosts (i.e., IP/hostname, username, and password) and its services need to be installed in each of them. This is done by defining which groups each host belongs to. The most important groups are control, network, compute, monitoring and storage. Kolla-Ansible will take care of installing the required services to each host depending on the groups they belong to. In order to match the hardware setup as described before, the multimode configuration looks like below:

```
 [control]
11.22.33.11 ansible_user=user_name
ansible_password=user_passwd ansible_become=true
[network]
11.22.33.22 ansible_user=user_name
ansible_password=user_passwd ansible_become=true
[compute]
11.22.33.44 ansible_user=user_name
ansible_password=user_passwd ansible_become=true
[monitoring]
#select the control
11.22.33.44
[storage]
localhost       ansible_connection=local
become=true
[deployment]
localhost       ansible_connection=local
become=true
```

The "globals.yml" configuration file is used for network configuration, OpenStack package definition, certification, repository, and storage assignment. According to the software setup defined previously, for the PoC of the server at the edge, it looks like below:

```
# You can use this file to override _any_ variable
throughout Kolla.
# Additional options can be found in the
# 'kolla-ansible/ansible/group_vars/all.yml' file.
Default value of all the
# commented parameters are shown here, To override
the default value uncomment
# the parameter and change its value.
###############
# Kolla options
###############
# Valid options are [ COPY_ONCE, COPY_ALWAYS ]
#config_strategy: "COPY_ALWAYS"
# Valid options are ['centos', 'debian',
'oraclelinux', 'rhel', 'ubuntu']
kolla_base_distro: "ubuntu"
# Valid options are [ binary, source ]
kolla_install_type: "source"
# Valid option is Docker repository tag
openstack_release: "rocky"
# Location of configuration overrides
#node_custom_config: "/etc/kolla/config"
# This should be a VIP, an unused IP on your
network that will float between
# the hosts running keepalived for high-
availability. If you want to run an
# All-In-One without haproxy and keepalived, you
can set enable_haproxy to no
# in "OpenStack options" section, and set this
value to the IP of your
# 'network_interface' as set in the Networking
section below.
kolla_internal_vip_address: "XX.YY.ZZ.VIP"
```

Where Kolla_internal_vip_address could be, for example, 11.22.33.99.

```
################
# Docker options
################
# Below is an example of a private repository with
authentication. Note the
# Docker registry password can also be set in the
passwords.yml file.
docker_registry: "XX.YY.ZZ.DEV:5000"
#docker_namespace: "regionone"
#docker_registry_username: "sam"
#docker_registry_password:
"correcthorsebatterystaple"
```

In our case, docker_registry is hosted on the development board, for example, 11.22.33.55.

```
##############################
# Neutron - Networking Options
##############################
# This interface is what all your api services will
be bound to by default.
# Additionally, all vxlan/tunnel and storage
network traffic will go over this
# interface by default. This interface must contain
an IPv4 address.
# It is possible for hosts to have non-matching
```

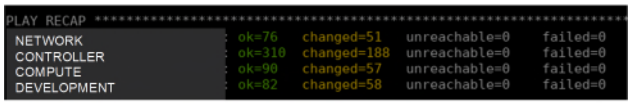
```
names of interfaces - these can
# be set in an inventory file per host or per group
or stored separately, see
#
http://docs.ansible.com/ansible/intro_inventory.htm
l
# Yet another way to workaround the naming problem
is to create a bond for the
# interface on all hosts and give the bond name
here. Similar strategy can be
# followed for other types of interfaces.
network_interface: "eno1.1"
# These can be adjusted for even more
customization. The default is the same as
# the 'network_interface'. These interfaces must
contain an IPv4 address.
#kolla_external_vip_interface: "{{
network_interface }}"
api_interface: "{{ network_interface }}"
#storage_interface: "{{ network_interface }}"
#cluster_interface: "{{ network_interface }}"
#tunnel_interface: "{{ network_interface }}"
#dns_interface: "{{ network_interface }}"
# This is the raw interface given to neutron as its
external network port. Even
# though an IP address can exist on this interface,
it will be unusable in most
# configurations. It is recommended this interface
not be configured with any IP
# addresses for that reason.
neutron_external_interface: "eno1.2"
# Valid options are [ openvswitch, linuxbridge,
vmware_nsxv, vmware_nsxv3, vmware_dvs, opendaylight
]
# if vmware_nsxv3 is selected, enable_openvswitch
MUST be set to "no" (default is yes)
#neutron_plugin_agent: "openvswitch"
# Valid options are [ internal, infoblox ]
#neutron_ipam_driver: "internal"
###################
# OpenStack options
###################
# Use these options to set the various log levels
across all OpenStack projects
# Valid options are [ True, False ]
#openstack_logging_debug: "False"
# Valid options are [ none, novnc, spice, rdp ]
#nova_console: "novnc"
# OpenStack services can be enabled or disabled
with these options
enable_cinder: "yes"
enable_cinder_backend_lvm: "yes"
enable_collectd: "yes"
enable_gnocchi: "yes"
enable_grafana: "yes"
enable_heat: "yes"
enable_horizon: "yes"
enable_horizon_magnum: "yes"
enable_horizon_tacker: "yes"
enable_horizon_zun: "yes"
enable_influxdb: "yes"
enable_kuryr: "yes"
enable_magnum: "yes"
enable_swift: "yes"
enable_telegraf: "yes"
enable_tacker: "yes"
enable_zun: "yes"
########################
# Glance - Image Options
########################
glance_backend_ceph: "no"
glance_backend_swift: "yes"
glance_enable_rolling_upgrade: "no"
################################
# Cinder - Block Storage Options
################################
cinder_backup_driver: "swift"
################################
# Swift - Object Storage Options
################################
swift_devices_match_mode: "strict"
swift_devices_name: "KOLLA_SWIFT_DATA"
```

IX. Deployment Sequence

Once the environment is ready, the deployment is straightforward. The very first time it is suggested to use the “pull” command before “deploy” to populate the local registry with the needed containers. In fact, the local registry is still empty. Pull will fail, pointing to the container still missing. The missing container could be easily added following the sequence:

```
docker pull
“this_openstack_component:rocky”
docker image tag
“this_openstack_component:rocky”
11.22.33.55:5000/this_openstack_componen
t:rocky
docker push
11.22.33.55:5000/this_openstack_componen
t
```

once the local repository is completed, the “pull” command becomes optional and deploy is possible using only three commands: bootstrap-servers, prechecks and deploy.



```
PLAY RECAP ************************************************************
 NETWORK        : ok=76   changed=51   unreachable=0   failed=0
 CONTROLLER     : ok=310  changed=188  unreachable=0   failed=0
 COMPUTE        : ok=90   changed=57   unreachable=0   failed=0
 DEVELOPMENT    : ok=82   changed=58   unreachable=0   failed=0
```

Figure 8. The Kolla-ansible deploy successfully result

```
sudo ./kolla-ansible -i multinode
bootstrap-servers
sudo ./kolla-ansible -i multinode
prechecks
(sudo ./kolla-ansible -i multinode pull)
sudo ./kolla-ansible -i multinode deploy
```

The result is shown in Figure 8.

After the deployment, a few commands are needed for the very first set up of the manager, like the definition of allowed volume size, basic test container image, and environment definitions.

```
./kolla-ansible -i multinode post-deploy
source /etc/kolla/admin-openrc.sh
./init-runonce
```

At the end of the sequence, the deployment has been done and the server is ready and can be used.

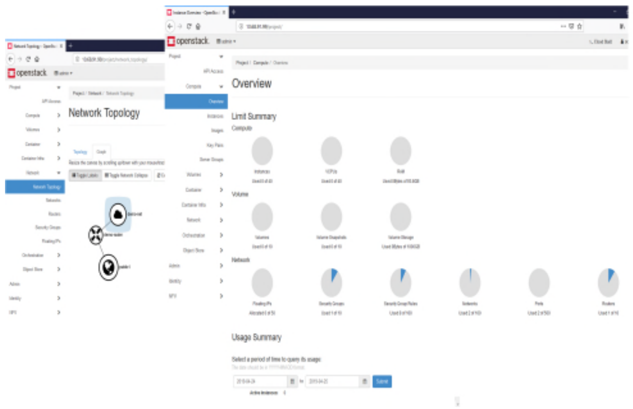


Figure 9. OpenStack Horizon Dashboard

Figure 9 shows the result of the deployment using the OpenStack standard dashboard (Horizon).

X. RESULT

It is interesting to analyze the distribution of the container executed by Kolla-Ansible deploy action. That investigation is useful to understand which and how resources are consumed by the infrastructure itself and so how heavy could be the cost of the SDN-NFV in the radio node. The "docker ps" is a command that could be used per any board to collect the list of running containers (see Figure 10).

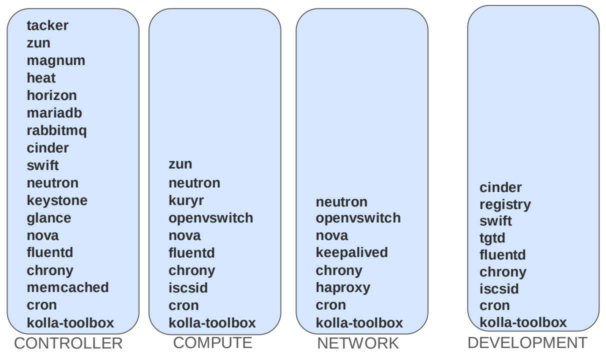


Figure 10. OpenStack Components distribution

The controller is the most populated board. This is not a problem. The experiment goal was to understand the minimal set of hardware resources needed by the Server at the edge of the network, but the Compute board usage is the most critical, since the Radio Interface boards could be connected to the node as "radio devote" compute board (as described in [3]). For that reason, this study is not focusing to the Controller or Network board resources usage, but to the Compute board. Central Processor Unit (CPU), RAM and Disk usage in compute board have been measured. The CPU load is normally less than 1%, disk usage is around 17GB (8% of available storage) and RAM usage is about 770 megabyte (MB) of the available 32GB. A comparison with minimal hardware requested by OPNFV is interesting:

TABLE II. COMPUTE BOARD MINIMAL HARDWARE REQUIREMENTS

Object	*Test Lab*	*OPNFV*
CPU socket	1	2
Disk (GB)	20	256
RAM (GB)	0,7	16

Managing and supervising containers is done using standard OpenStack Horizon and Grafana Dashboards.

XI. CONCLUSION

Working directly with OpenStack components instead of using an SDN-NFV package distribution, like OPNFV, allowed us to have a better idea of the minimal hardware resources setup at the cost of the maintenance. It is not so simple to verify the compatibility between different OpenStack components versions and which patches/modifications might be needed. The availability and correctness of Opensource documentations, in our point of view, need to improve. Components cross-reference dependencies, patches and exception descriptions are not always easy to find and clear. Whenever available, most of the documentation referred to old Linux/OpenStack releases. It looks like different projects are moving forward totally independently from each other, with the result that document begins to be obsolete within 6 months and the interplay among different components needs plenty of documentation review and corrections. This impacted on the extended time needed to achieve a correct set up of the software environment in the experimental study, and that is surely not an optimal condition. If the cost of the maintenance is so relevant, then integrated SDN-NFV distributions have their own meaning from a product point of view. Yet, they are too expensive in terms of the minimal set of hardware resource requirement. However, components like ONAP, O-RAN or the latest delivery of Kubernetes well integrated with the NOS have a huge value. This results probably in enough reasons for considering SDN-NFV integrated distributions while searching for a more product-oriented solution. The Opensource community should try to have a minimal distribution package for the SDN-NFV integrated distribution, more careful to the value of the hardware resources availability. Indeed, this will be a key factor to use SDN-NFV very close to the End User.

REFERENCES

[1] Ericsson, "Ericsson Mobility Report," Jun 2019, Rev. A, available from https://www.ericsson.com/en/mobility-report/ reports/june-2019 [retrieved: Sep, 2019].

[2] M. Branda, "Acceleration of the 5G NR global standards gains industry momentum," Sep, 2016, OnQ Blog, Qualcomm, available from https://www.qualcomm.com/news/onq/2016/09/ 27/acceleration-5g-nr-global-standard-gains-industry-momentum [retrieved: Sep, 2019].

[3] C. Vitucci and A. Larsson, "Flexible 5G Edge Server for Multi Industry Service Network," International Journal on Advances in Networks and Services, Vol. 10, no. 3-4, 2017, pp. 55.65, ISSN: 1942-2644.

[4] T. Cucinotta, L. Abeni, M. Marinoni and C. Vitucci, "The importance of being OS-aware in Performance Aspects of Cloud Computing Research," in Proceedings of the 8th International Conference on

Cloud Computing and Services Science (CLOSER 2018), Mar, 2018, pp. 626-633

[5] 5GPPP Architecture Working Group, “View on 5G Architecture,” version 2.0, December 2017, available from https://5g-ppp.eu/wp-content/uploads/2018/01/5G-PPP-5G-Architecture-White-Paper-Jan-2018-v2.0.pdf [retrieved: Nov, 2018].

[6] ETSI paper, “Network Functions Virtualisation (NFV); Use Cases,” ETSI GS NFV 001, v.1.1.1, 2013, available from https://www.etsi.org/deliver/etsi_gs/nfv/001_099/001/01.01.01_60/gs_nfv001v010101p.pdf [retrieved: Sep, 2019]

[7] OpenStack Foundation, OpenStack Community Software Reference Page, available from https://www.openstack.org/software/ [retrieved: Sep, 2019]

[8] R. Vilalta, A. Mayoral, R. Casellas, R. Martínez and R. Muñoz, “Experimental demonstration of distributed multi-tenant cloud/fog and heterogeneous SDN/NFV orchestration for 5G services,” 2016 European Conference on Networks and Communications (EuCNC), Athens, 2016, pp. 52-56

[9] H. M. Abdel-Atty, R. S. Alhumaima, S. M. Abuelenin and E. A. Anowr, “Performance Analysis of Fog-Based Radio Access Networks,” in IEEE Access, vol. 7, 2019, pp. 106195-106203.

[10] OPNFV Licensed webpage, “OPNFV Fuel Installation instruction”, chapter 2.4 Hardware Requirements, available from https://opnfv-fuel.readthedocs.io/en/latest/release/installation/installation.instruction.html [retrieved; Jan, 2020]

[11] Canonical Ltd. Ubuntu, Ubuntu reference page, available from https://ubuntu.com/ [retrieved: Sep, 2019]

[12] Ismail Baydan, “Apt ad Apt-Get Tutorial With Examples”, Poftut online tutorial, latest update by Nov. 2019, available from https://www.poftut.com/apt-and-apt-get-tutorial-with-examples/ [retrieved: Jan, 2020]

[13] OPNFV Project a Series of LF Projects, OPNFV reference page, available from https://www.opnfv.org/ [retrieved: Jan, 2020]

[14] ONAP a Series of LF Projects, ONAP reference page, available from https://www.onap.org/ [retrieved: Jan, 2020]

[15] O-RAN Alliance e.V., “Operator Defined Next Generation RAN Architecture and Interfaces”, 2019, available from https://www.o-ran.org/ [retrieved: Jan, 2020]

[16] OpenStack Foundation, “Container Optimized sample configuration”, available from https://www.openstack.org/software/sample-configs/#container-optimized [retrieved: Jan, 2020]

[17] OpenStack Foundation, “Deploys OpenStack in Containers using Ansible”, available from https://www.openstack.org/software/releases/stein/components/kolla-ansible [retrieved: Jan, 2020]

[18] OpenStack Foundation, “Quick Start how to deploy OpenStack using Kolla and Kolla-Ansible on bare metal servers or virtual machine”, available from https://docs.openstack.org/kolla-ansible/ocata/user/quickstart.html [retrieved: Jan, 2020]