Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Application of SDN for the OpenStack-based NFV requirements (5G introduction)

27.02.2020

An aim of this project is to set up a working OpenStack environment, acknowledge with different method of installation and services that are part of OpenStack and set up simple Network with one instance as a server and some as clients.

Introduction: What exactly is OpenStack and what its services are responsible for?

OpenStack is a set of software tools for building and managing cloud computing platforms for public and private clouds. Backed by some of the biggest companies in software development and hosting, as well as thousands of individual community members, many think that OpenStack is the future of cloud computing. OpenStack is managed by the OpenStack Foundation, a non-profit that oversees both development and community-building around the project.

There are many components OpenStack consists of, but the community has identified nine "core" services. Here I want to familiarize you with them.

1) **Nova** is the primary computing engine behind OpenStack. It is used for deploying and managing large numbers of virtual machines
2) **Swift** is a storage system for objects and files. Rather than the traditional idea of a referring to files by their location on a disk drive, developers can instead refer to a unique identifier referring to the file or piece of information and let OpenStack decide where to store this information. This makes scaling easy.
3) **Cinder** is a block storage component, which is more analogous to the traditional notion of a computer being able to access specific locations on a disk drive. This might be important when data access speed is needed.
4) **Neutron** provides the networking capability for OpenStack. It helps to ensure that each of the components of an OpenStack deployment can communicate with one another.
5) **Horizon** is the dashboard behind OpenStack. It is the only graphical interface to OpenStack.
6) **Keystone** provides identity services for OpenStack. It is essentially a central list of all of the users of the OpenStack cloud, mapped against all of the services provided by the cloud, which they have permission to use.
7) **Glance** provides image services to OpenStack. In this case, "images" refers to images (or virtual copies) of hard disks.
8) **Ceilometer** provides telemetry services, which allow the cloud to provide billing services to individual users of the cloud.
9) **Heat** is the orchestration component of OpenStack, which allows developers to store the requirements of a cloud application in a file that defines what resources are necessary for that application.

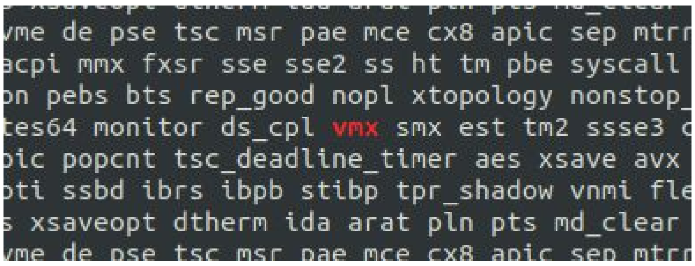
Part 1: Setting up an environment

As my desktop OS I chose Ubuntu 18.04 LTS "Bionic Beaver". It is second latest edition yet more tested and with more community bug support.
Eventually, I ended up with Win10 and Ubuntu 18.04 dualboot.

It is very important to enable virtualisation of your CPU in BIOS. You can simply check if your CPU supports virtualisation by running below command in linux terminal:

For Intel: `grep --color vmx /proc/cpuinfo`
For AMD: `grep --color svm /proc/cpuinfo`

As an output you should see vmx/svm in red font:



```
vme de pse tsc msr pae mce cx8 apic sep mtrr
acpi mmx fxsr sse sse2 ss ht tm pbe syscall
on pebs bts rep_good nopl xtopology nonstop_
tes64 monitor ds_cpl vmx smx est tm2 ssse3 c
pic popcnt tsc_deadline_timer aes xsave avx
pti ssbd ibrs ibpb stibp tpr_shadow vnmi fle
s xsaveopt dtherm ida arat pln pts md_clear
vme de pse tsc msr pae mce cx8 apic sep mtrr
```

Next step is to choose a hypervisor for x86 virtualization. There are plenty of software you can choose from. For example KVM or VirtualBox. Both are good for our purpose. My bet is Oracle VirtualBox.

Link: https://www.virtualbox.org/wiki/Linux_Downloads

Part 2: DevStack!

There are many of installation methods of OpenStack. Some of them are strictly for development use, some are more complex and used for production.

DevStack is a series of extensible scripts used to quickly bring up a complete OpenStack environment.
It is applied as a development environment and as the basis for much of the OpenStack project's functional testing .
DevStack is created with a view to to support a large number of configuration options and alternative platforms and support services.

The list of what is really supported by Devstack:

- BaseOS - Ubuntu, Fedora, RHEL/CentOS
- Database – MySQL
- Queues – Rabbit
- Web Server – Apache
- OpenStack Network – Neutron

The default services configured by DevStack:

- Networking
- Dashboard
- Identity
- Object Storage
- Image Service
- Block Storage
- Compute
- Placement

Additional services can be added by using the plugins plugin mechanism to call scripts that perform the configuration and startup of the service.

all information from:
https://docs.openstack.org/devstack/latest/index.html

TripleO is an Opensource project developed to facilitate and improve the installation of OpenStack and its subsequent management. This the most common way of setting up OpenStack used by corporations due to its most customizable nature.

The options provided by TripleO

- Provisioning of bare-metal servers
- Deploying on already provisioned servers(split-stack)
- Deploying on already provisioned servers(split-stack)
- Containerized or non-containerized OpenStack deployments
- Custom roles and networks, spine-leaf topology
- Integration with 3rd party backends (Cinder, Neutron, Manila)
- SSL/IPSEC internal traffic encryption
- controlling node placement and IP assignment
- Configuring High Availability for the control plane and workloads

Kolla's mission is to provide production-ready containers and deployment tools for operating OpenStack clouds. It is very friendly for users with little experience, as it allows for quick implementation of openstack. In later stages, it is possible to configure the cloud, adjusting it to the user's needs.

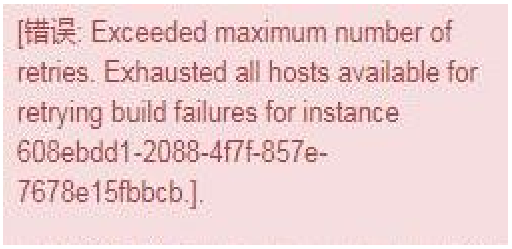
- Distro choice of CentOS, Oracle, Linux, RHEL, UBUNTU
- Build from packages or build from source
- Intent is to deploy the big tent at 100 node scale
- Docker containers for atomic upgrades
- Full customization of the OpenStack available

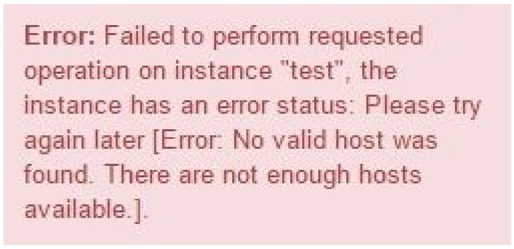
The first but, as said before, least stable installation of Openstack I will cover is DevStack. Installation is pretty simple.

I hardly recommend using fresh installation of Ubuntu 18.04 LTS. From my experience, Ubuntu 16.04, despite being the most tested, is not adapted enough for DevStack and many errors will occur during the process of deploy and some of them are really, REALLY unintuitive. Just respect your time and switch to 18.04
You have to set up a single virtual machine that fulfills below requirements:

At least 2 core processor
At least 8GB RAM
At least 60GB hard drive

Why “at least”? These are the minimum requirements that will ensure the process of installation will go smooth and the dashboard work with some decent speed. Yes, you would be able to run some “low-end” in instances, but you could encounter one of these errors pretty soon:


[错误: Exceeded maximum number of retries. Exhausted all hosts available for retrying build failures for instance 608ebdd1-2088-4f7f-857e-7678e15fbbcb.].


Error: Failed to perform requested operation on instance "test", the instance has an error status: Please try again later [Error: No valid host was found. There are not enough hosts available.].

Remember, VM’s are able to share a single CPU, but not RAM!
After you installed the VM, connect to it via SSH and proceed with the instructions below.

1) You’ll need to run Devstack as a non-root user with sudo enabled. Create new user named “stack”:
```
sudo useradd -s /bin/bash -d /opt/stack -m stack
```

2) Because “stack” will be making many changes in the system, you should add sudo privileges and disable the password:
```
echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee
/etc/sudoers.d/stack
```

3) Switch to that user:
```
sudo su - stack
```

4) install git and clone DevStack:
```
sudo apt-get install git
git clone https://github.com/openstack-dev/devstack.git -b
stable/pike devstac
```

   “Pike” stands for the version of openstack you want to install. The first letters of different versions are in the alphabetical order. For the current state, the newest release is “Train” and “Ussurl” is under development.

5) Change to the devstack directory:

```
cd devstack/
```

6) Check your host ip (could be done with `ip a` or `ifconfig) and` create the local.conf file with 4 passwords and your host IP. This one is the minimum config required to successfully run DevStack.
[[local|localrc]]
```
ADMIN_PASSWORD=secret
DATABASE_PASSWORD=\$ADMIN_PASSWORD
RABBIT_PASSWORD=\$ADMIN_PASSWORD
SERVICE_PASSWORD=\$ADMIN_PASSWORD
HOST_IP=YOUR.HOST.IP.ADDR
RECLONE=yes
```

7) All that remain is to run bash script!
```
./stack.sh
```

If you're lucky, after 20-30 minutes you should see an output like this:
```
This is your host IP address: x.x.x.x
This is your host IPv6 address: ::1
Horizon is now available at http://x.x.x.x/dashboard
Keystone is serving at http://x.x.x.x/identity/
The default users are: admin and demo
The password: somefancyrandompassword
DevStack Version: pike
OS Version: Ubuntu 18.04 bionic
stack.sh completed in "a lot" seconds.
```

In this case, there is nothing more to do than logging into the dashboard and have fun using OpenStack GUI (or CLI, if you want) until the next reboot.

If you are, like me, not lucky enough, you will encounter some world dumps.
If the terminal is so kind to tell to "die" (`[Call Trace] ./stack.sh:137:die`) there is a simple workaround:
In the local.conf add one line `FORCE=yes`. Then run `./unstack.sh` and `./stack.sh` again.

In case there are some unexpected world dumps without tracebacks given, make sure your VM have enough free space available.

To completely clean devstack installation filer run `./clean.sh` and `rm -rf` all remaining files.

Part 3: Kolla-Ansible: VM

The second and in my opinion the most optimal method of setting up an OpenStack is using Kolla-Ansible. It is definitely more complex than DevStack, but much more customizable and reliable. It supports multi-node installation thus can be used in production environment.

Unfortunately, my PC does not have enough resources to set up three nodes (hypervisor and two compute nodes) so I decided to set up all-in-one (one VM to rule them all).

Due to the many steps I will not number steps in this documentation. Let's start then.

First of all you have to create one virtual machine. Like in the previous part, VirtualBox is the chosen one of mine, but KVM/QEMU would also do the job.

The minimal requirements are:
4 vCPUs
8GB RAM
100GB Disk space

I hardly suggest to give more resources if you want to use OpenStack without any problems.

I have observed that Openstack uses circe 6GB of RAM (maybe more) by itself, so every additional RAM you allocate for the VM will be used as a part of instances flavours. For example: I gave 8192MB of RAM for my VM. As I set up OpenStack and it came to creating instances, I was only able to create one 1GB RAM "server" and 1 or 2 small 512MB RAM "client" instances (depends of the number of processes running in the background).

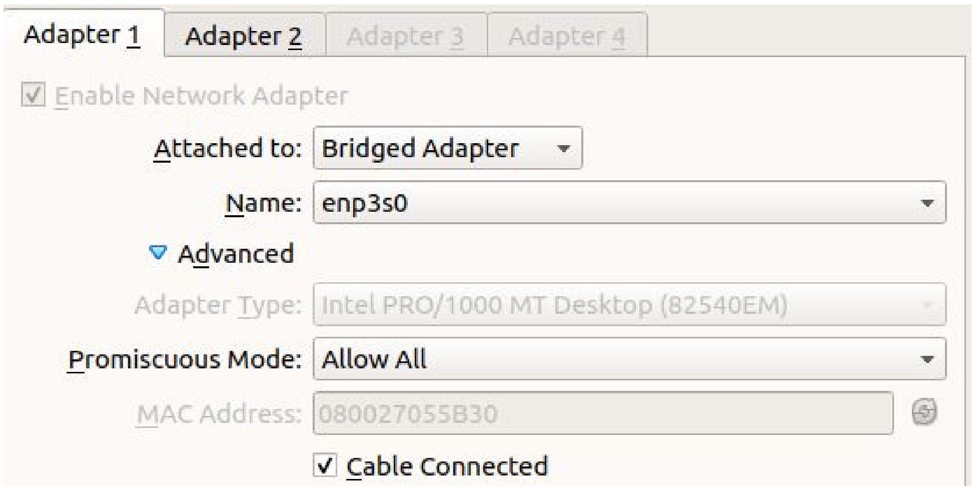
It is time to choose an OS for that VM. It is allowed to use CentOS, RHEL (RedHat Enterprise Linux) or Ubuntu. RHEL is a commercial distro of CentOS supported by Red Hat. It was made strictly for use with OpenStack. Nevertheless, it have one big con: a subscription for Red Hat is needed to install repos in that particular OS.

CentOS and Ubuntu Server are equally a good choice, but as I am more familiar with Ubuntu, I chose that one.

Kolla can be used with any version >= 14.04. It is always suggested to use the newest version, but there is one trick. Ubuntu 18 has switched to Netplan for configuring network interfaces. Netplan is based on YAML based configuration system. As I have always be using oldschool `/etc/network/interfaces`, I stood one step behind the fresh distro and use 16.04 Server instead.

Download link: http://pl.releases.ubuntu.com/16.04/ubuntu-16.04.6-server-amd64.iso

It is very important to add the second Network Interface (NIC). On both set "Bridged Adapter", connect it to your primary interface (i.e. enp3s0) and in Advanced settings set Promiscuous mode to "Allow all". It will allow us to pass all traffic it receives to the central processing unit rather than passing only the frames that the controller is specifically programmed to receive.


Adapter 1 | Adapter 2 | Adapter 3 | Adapter 4

[✓] Enable Network Adapter

Attached to: Bridged Adapter

Name: enp3s0

Advanced

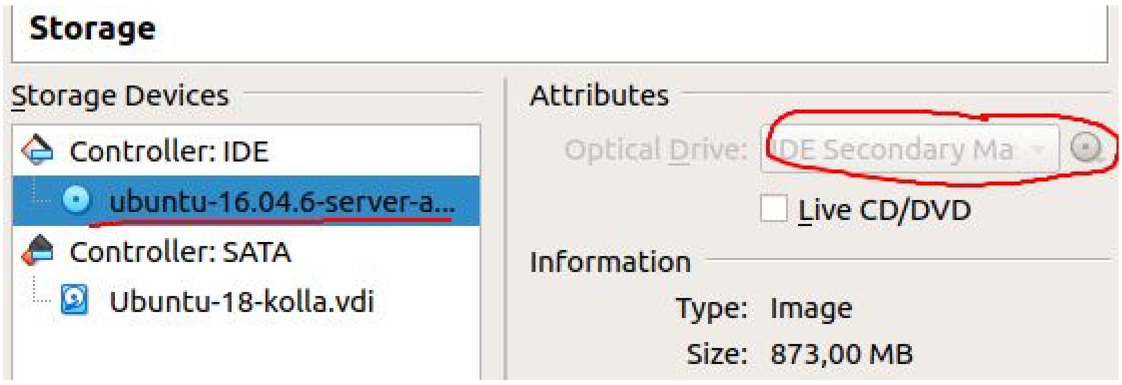
Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

Promiscuous Mode: Allow All

MAC Address: 080027055B30

[✓] Cable Connected

Please be sure to load the ubuntu .iso file to optical drive. To do so, click that CD icon on the right and select the wanted .iso.


Storage

Storage Devices

- Controller: IDE
  - ubuntu-16.04.6-server-a...
- Controller: SATA
  - Ubuntu-18-kolla.vdi

Attributes

Optical Drive: IDE Secondary Ma

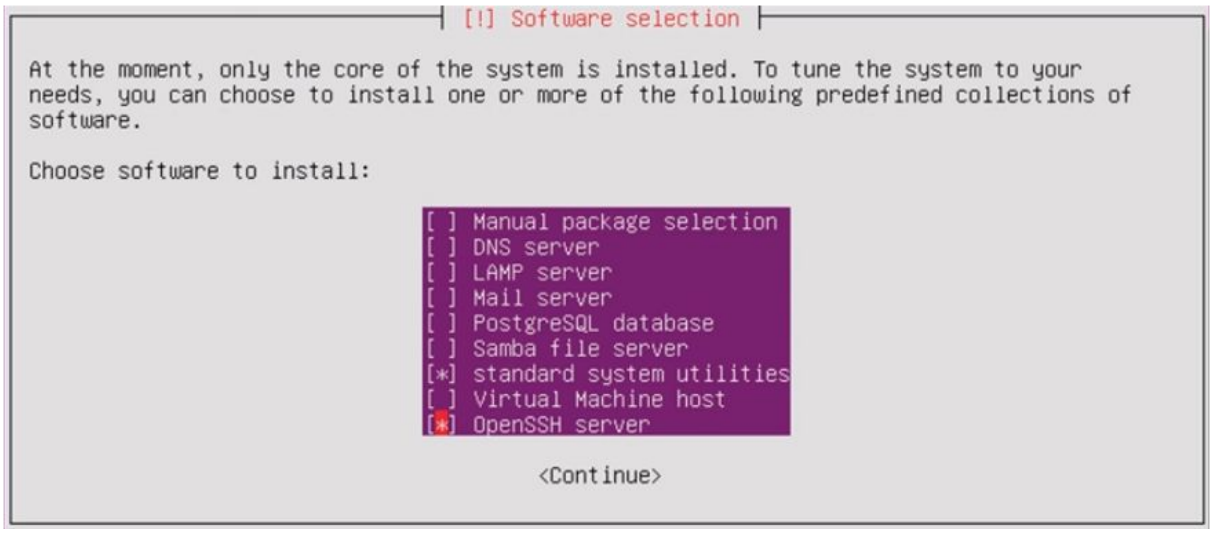
[ ] Live CD/DVD

Information

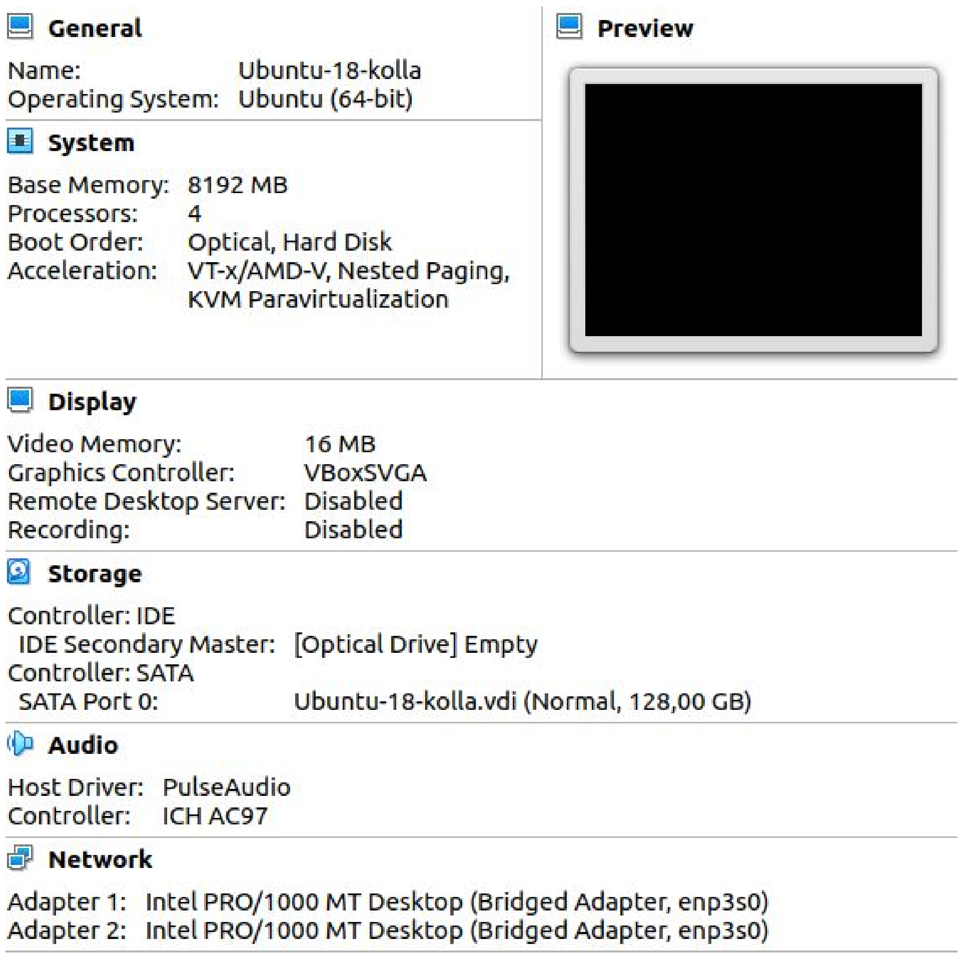
Type: Image

Size: 873,00 MB

During the installation process just skip everything else than setting hostname, user, password and additional services. There the only additional service we want to download during installation is OpenSSH. Check its box and go on.


```
[!] Software selection

At the moment, only the core of the system is installed. To tune the system to your
needs, you can choose to install one or more of the following predefined collections of
software.

Choose software to install:

                    [ ] Manual package selection
                    [ ] DNS server
                    [ ] LAMP server
                    [ ] Mail server
                    [ ] PostgreSQL database
                    [ ] Samba file server
                    [*] standard system utilities
                    [ ] Virtual Machine host
                    [*] OpenSSH server

                               <Continue>
```

The final result after installation will look more or less like this one below:


General

Name: Ubuntu-18-kolla
Operating System: Ubuntu (64-bit)

System

Base Memory: 8192 MB
Processors: 4
Boot Order: Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Preview

Display

Video Memory: 16 MB
Graphics Controller: VBoxSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage

Controller: IDE
IDE Secondary Master: [Optical Drive] Empty
Controller: SATA
SATA Port 0: Ubuntu-18-kolla.vdi (Normal, 128,00 GB)

Audio

Host Driver: PulseAudio
Controller: ICH AC97

Network

Adapter 1: Intel PRO/1000 MT Desktop (Bridged Adapter, enp3s0)
Adapter 2: Intel PRO/1000 MT Desktop (Bridged Adapter, enp3s0)

Part 4: Kolla-Ansible: Network

After all these steps, our environment is ready! Let's start configuring our network.
First of all type $ `ip a / ifconfig` in VM ic case to get your ip. With that number, we can connect to that VM via ssh. It will make our work much easier.

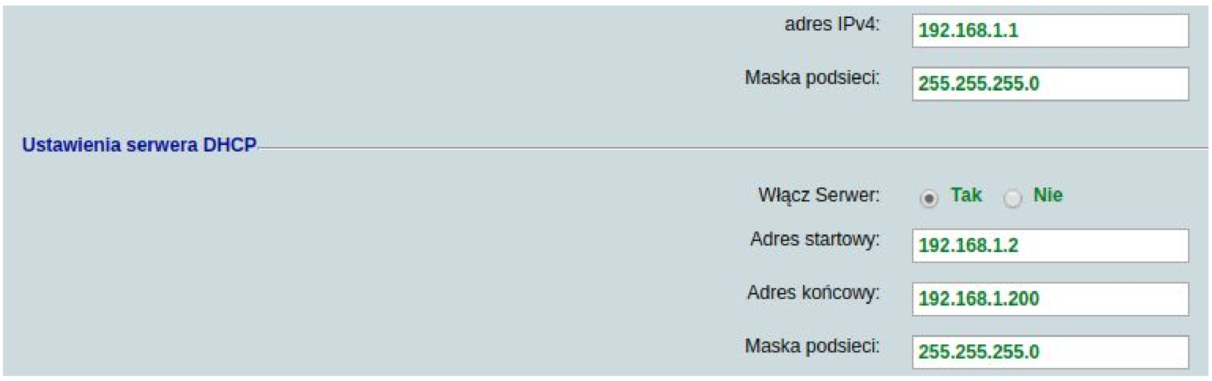
```
ssh username@x.x.x.x
```

In my case: `ssh kolla@192.168.1.210` (my address is after configuring the interfaces)

Next, run $ `sudo apt update && sudo apt upgrade -y`
-y is to bypass the "are you sure you want to install…" prompts.

Now it is time for us to set up a static IP for one VM's interface and manual for the second. Second interface is for neutron external network, this interface will be used by OpenvSwitch and will be attached to br-ex. It should be in `UP`state without given address.

You will need some free IPs from your networks, so log in to router and configure your DHCP pool.



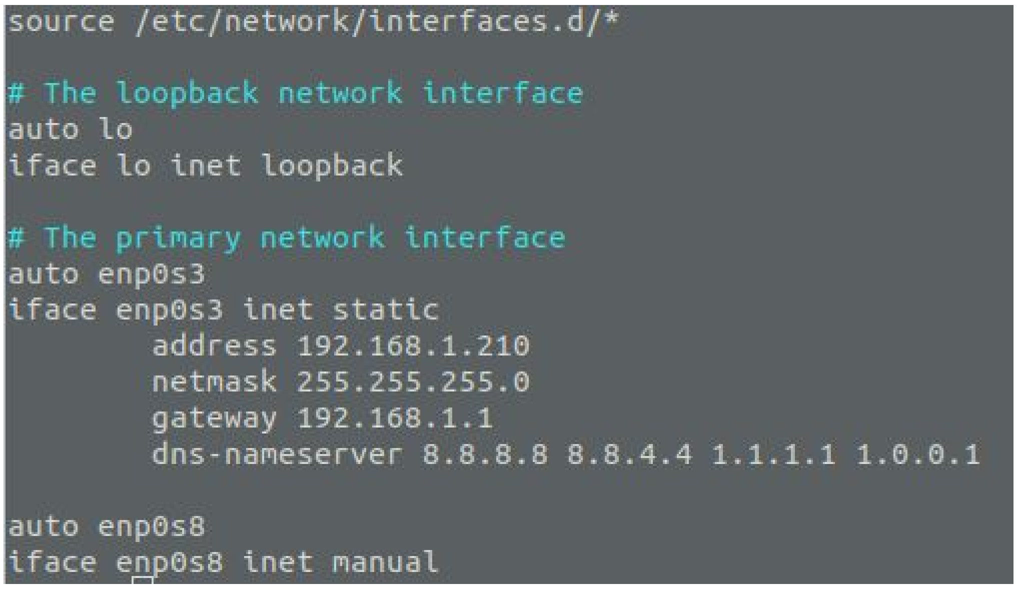
adres IPv4:	192.168.1.1
Maska podsieci:	255.255.255.0

Ustawienia serwera DHCP

Włącz Serwer:	[✓ Tak] Nie
Adres startowy:	192.168.1.2
Adres końcowy:	192.168.1.200
Maska podsieci:	255.255.255.0

My gateway is 192.168.1.1, so I set DHCP address range from 192.168.1.2 to 192.168.1.200 to have up to 54 free IPs

Then, $ `sudo vi /etc/network/interfaces` and type as follow:



```
source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto enp0s3
iface enp0s3 inet static
        address 192.168.1.210
        netmask 255.255.255.0
        gateway 192.168.1.1
        dns-nameserver 8.8.8.8 8.8.4.4 1.1.1.1 1.0.0.1

auto enp0s8
iface enp0s8 inet manual
```

My DNS servers are Google, Google backup address and one open source Cloudflare DNS with its backup.

Save, quit and restart networking with the command: `sudo systemctl restart networking`. After that our main interface will have two different IPs. To solve this problem simply reboot VM an ssh using new address (mine is 192.168.1.210).

Part 5: Kolla-Ansible: Installing packages

In this part we will install all required packages for our Kolla. It is really tricky part, but I will cover that later.

First of all we have to install Jinja2, python-pip and libssl.
What exactly are they?

Jinja is a web template engine for the Python programming language and is licensed under a BSD License created by Armin Ronacher. It is similar to the Django template engine but provides Python-like expressions.
pip is just a system used to install and manage software packages written in python.
libssl-dev is a package that consists of development libraries, header files, manpages for libcrypto and libssl. It is part of the OpenSSL project's implementation of the SSL and TLS cryptographic protocols
~Wikipedia

```
$ sudo apt install python-jinja2 python-pip libssl-dev
```

Then upgrade pip:
```
$ sudo -H pip install -U pip
```
If an error occurs, just run
```
$ export LC_ALL=C
```

Installation of our automatization environment:
```
$ sudo -H pip install -U ansible
```

A short interruption: **Ansible** is an open-source software provisioning, configuration management, and application-deployment tool. It runs on many Unix-like systems, and can configure both Unix-like systems as well as Microsoft Windows. It includes its own declarative language to describe system configuration.
~Wikipedia

And installation of the kolla scripts ver. 6.0.0.0rc2 itself:
```
$ sudo -H pip install kolla-ansible==6.0.0.0rc2
```

Copy some files (-r to make directory if one does not exist):
```
$ sudo cp -r /usr/local/share/kolla-ansible/etc_examples/kolla
/etc/kolla
```

Part 6: Kolla-Ansible: Configuring environment

Open the configuration file:
```
$ sudo vi /etc/kolla/globals.yml
```
Here we have all configuration of our openstack. We can base only on this file to configure our all-in-one environment. Due to its length I will show you only the lines I have changed or that are non-default.

```
kolla_base_distro: "centos"   #can be RHEL, Ubuntu, CentOS - no difference at all
kolla_install_type: "binary"
openstack_release: "queens"  #my favourite :)
node_custom_config: "/etc/kolla/config"  #overloading kolla configuration
kolla_internal_vip_address: "192.168.1.211"  #communication between
modules
kolla_external_vip_address: "192.168.1.212"  #dashboard address
```

`network_interface: "enp0s3"` #primary interface where all api services will be bound to by default

```
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 }}"
```

`neutron_external_interface: "enp0s8"` #internal communication between services CHECK YOUR INTERFACE NAME BY IP A COMMAND!

`neutron_plugin_agent: "openvswitch" #`virtual tool to create switches

`kolla_enable_tls_external: "yes" #`load certificates to kolla

`nova_compute_virt_type: "qemu"` #because we are not in kvm, qemu is an emulator, not a virtualizer

Now we have to generate ssh keys for users kolla and root:
$ `ssh-keygen` (no password, just press enter)

```
$ sudo su -
# ssh-keygen
```

Let's make new kolla directory:

```
$ sudo mkdir -p /etc/kolla/config/nova
$ sudo vi /etc/kolla/config/nova/nova-compute.conf
```

We are running Openstack on a VM, so we have to make this file where we can overload some nova config to fit kolla to the virtenv:

```
[libvirt]
virt-type-qemu
cup_mode = none
```

Part 7: Kolla-Ansible: Running kolla yaml scripts

Now, we have came to the glorious times when we have to run kolla yamls to finish configuration and finally deploy the whole platform.

First of all, we must generate certificate:

```
$ sudo kolla-ansible certificates
```

We can check if they really exist by running:

```
$ ll /etc/kolla/certificates/private
```

There should be haproxy.crt and haproxy.key (HA stands for High Availability)

Generating passwords for openstack services

```
$ sudo kolla-genpwd
```

Passwords are stored at /etc/kolla/passwords.yml

Okay, now we are getting into the real business. We have to prepare our server by running:

```
$ sudo cp
/usr/local/share/kolla-ansible/ansible/inventory/all-in-one . #copy aio to local path
$ sudo kolla-ansible -i ./all-in-one bootstrap-servers
```

The next thing we have to do it to pull images for docker. This process is long, so we can go grab a beer, watch one episode of Star Wars: The Clone Wars and after that all the things should be done.

```
$ sudo kolla-ansible -i ./all-in-one pull
```

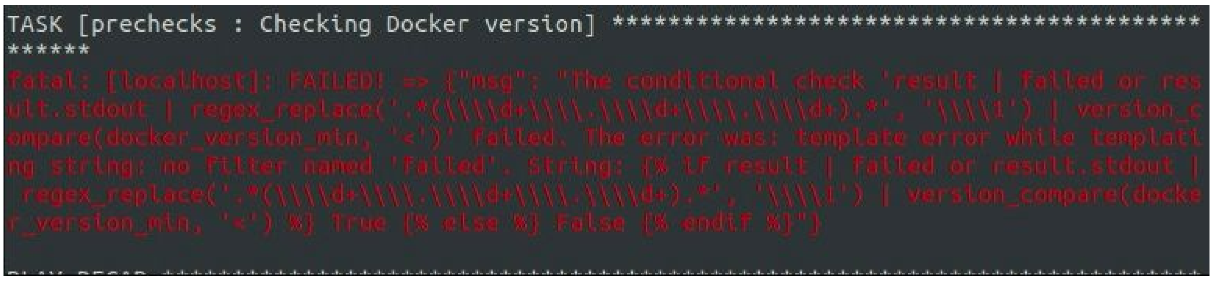
We can see all of the downloaded images by running:

```
$ sudo docker images
```

The last thing before deploy is to run precheck. This is where the problems might occur.

```
$ sudo kolla-ansible prechecks -i ./all-in-one
```

Aaaaaaaand…

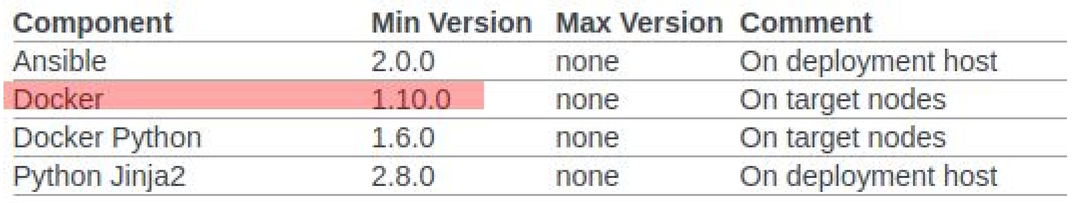


```
TASK [prechecks : Checking Docker version] ********************************************
******
fatal: [localhost]: FAILED! => {"msg": "The conditional check 'result | failed or res
ult.stdout | regex_replace('.*(\\\\d+\\\\.\\\\d+\\\\.\\\\d+).*', '\\\\1') | version_c
ompare(docker_version_min, '<')' failed. The error was: template error while templati
ng string: no filter named 'failed'. String: {% if result | failed or result.stdout |
 regex_replace('.*(\\\\d+\\\\.\\\\d+\\\\.\\\\d+).*', '\\\\1') | version_compare(docke
r_version_min, '<') %} True {% else %} False {% endif %}"}
```

As that error in labeled as “Docker version”, the first thing I have thought about is to check Docker version.

```
$ sudo docker version
```

In output it was written that mine version was 1.9.x. Unfortunately I do not remember exactly which one. I have started looking for the documentation for kolla-ansible to check that is the minimum supported Docker version. I have found that one:



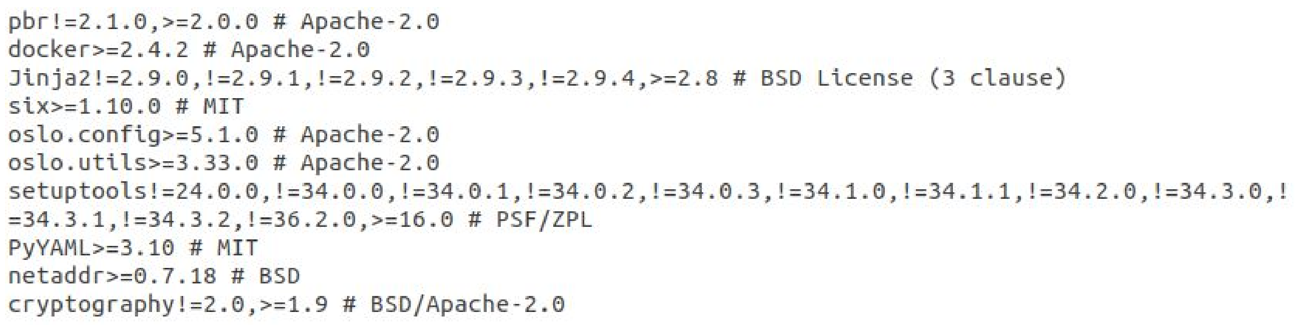
Component	Min Version	Max Version	Comment
Ansible	2.0.0	none	On deployment host
Docker	1.10.0	none	On target nodes
Docker Python	1.6.0	none	On target nodes
Python Jinja2	2.8.0	none	On deployment host

In order to upgrade Docker to the 1.10.0 (the latest version is not always the best one) I have followed that instructions:
https://docs.docker.com/install/linux/docker-ce/ubuntu/

When upgraded, I have ran precheck once again, that error appeared again. Despite the upgrade, after precheck docker version still was 1.9.X.

I have asked on a linux forum, where I was told to upgrade Docker using pip. So did I. Nevertheless, it did not change anything.
As I saw that documentation was for an older kolla-ansible, I started looking for the newer.

I gave up, I have found my particular kolla-ansible version in .zip and I downloaded it witch hope to find some readme, requirements files or to see strictly in YAML files which version is minimal.
Thanks God, I have found the Holy Graal. requirements.txt



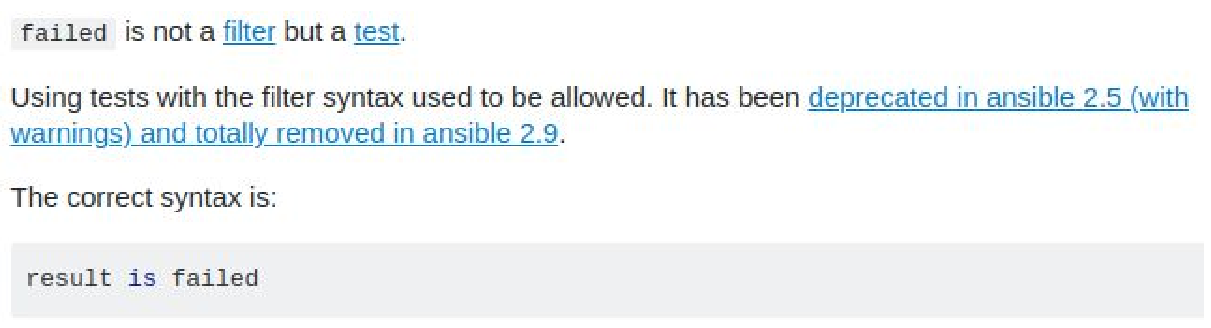
```
pbr!=2.1.0,>=2.0.0 # Apache-2.0
docker>=2.4.2 # Apache-2.0
Jinja2!=2.9.0,!=2.9.1,!=2.9.2,!=2.9.3,!=2.9.4,>=2.8 # BSD License (3 clause)
six>=1.10.0 # MIT
oslo.config>=5.1.0 # Apache-2.0
oslo.utils>=3.33.0 # Apache-2.0
setuptools!=24.0.0,!=34.0.0,!=34.0.1,!=34.0.2,!=34.0.3,!=34.1.0,!=34.1.1,!=34.2.0,!=34.3.0,!
=34.3.1,!=34.3.2,!=36.2.0,>=16.0 # PSF/ZPL
PyYAML>=3.10 # MIT
netaddr>=0.7.18 # BSD
cryptography!=2.0,>=1.9 # BSD/Apache-2.0
```

BUT
Guess what :)
Upgrading to this version does not help at all :)

The next step was to open a thread on stackoverflow. The link is below:
https://stackoverflow.com/questions/59832284/kolla-ansible-openstack-docker-version-failure/59842505#59842505

I have managed to uninstall all Docker files and install it from scratch. It was more or less helpful, because after precheck Docker version remained upgraded, but this did not fix the problem.

But, a user named Zeitounator gave me a clue:



`failed` is not a filter but a test.

Using tests with the filter syntax used to be allowed. It has been deprecated in ansible 2.5 (with warnings) and totally removed in ansible 2.9.

The correct syntax is:

```
result is failed
```

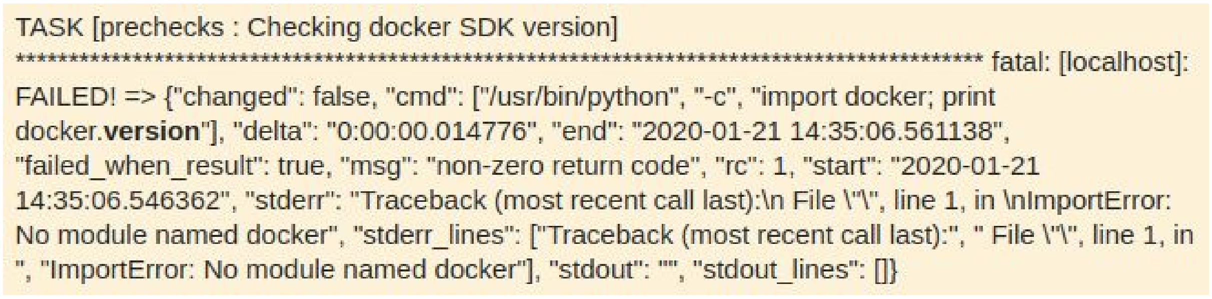
When I have run:

```
$ sudo ansible --version
```

It gave me an output of “2.9.6”. Victory. Now:

```
$ sudo -H pip install ansible==2.5.6
```

One command to rule them all. An Übercommand.
But life can't be too beautiful. After a few seconds after an another error popped up:



```
TASK [prechecks : Checking docker SDK version]
****************************************************************************************** fatal: [localhost]:
FAILED! => {"changed": false, "cmd": ["/usr/bin/python", "-c", "import docker; print
docker.version"], "delta": "0:00:00.014776", "end": "2020-01-21 14:35:06.561138",
"failed_when_result": true, "msg": "non-zero return code", "rc": 1, "start": "2020-01-21
14:35:06.546362", "stderr": "Traceback (most recent call last):\n File \"\", line 1, in \nImportError:
No module named docker", "stderr_lines": ["Traceback (most recent call last):", " File \"\", line 1, in
", "ImportError: No module named docker"], "stdout": "", "stdout_lines": []}
```

As it told me what was wrong, the solution was pretty simple:

```
$ sudo pip install docker
```

Run precheck once again and the whole script completed without errors. Success.

Now, the (pre)last thing: deploy:

```
$ sudo kolla-ansible deploy -i ./all-in-one
```

This one sets up containers, configures them with kolla-ansible scripts.

To check containers:

```
$ sudo docker ps -a
```

We have to generate the config file openrc:

```
$ sudo kolla-ansible -i ./all-in-one post-deploy
```

After that, we have to install OpenStack CLI with pip. This one will allow us to configure OpenStack through the command-line:

```
$ sudo -H pip install -U python-openstackclient
```

Then, we have to edit init-runonce. That file sets up our initial OpenStack features i.e. flavours, networks, routers. In my case, I have set up some limits of the maximum number of VCPUs, RAM etc. We have to change our network addresses also.

```
$ sudo vi /usr/local/share/kolla-ansible/init-runonce
```

My configuration:

```
EXT_NET_CIDR='192.168.1.0/24'
EXT_NET_RANGE='start=192.168.1.220,end=192.168.1.250'
EXT_NET_GATEWAY='192.168.1.1'

# 4 instances
openstack quota set --instances 4 ${ADMIN_PROJECT_ID}
```

```
# 4 cores
openstack quota set --cores 4 ${ADMIN_PROJECT_ID}
# 8GB ram
openstack quota set --ram 8192 ${ADMIN_PROJECT_ID}
Then:
$ sudo su -
# source /etc/kolla/admin-openrc.sh
```

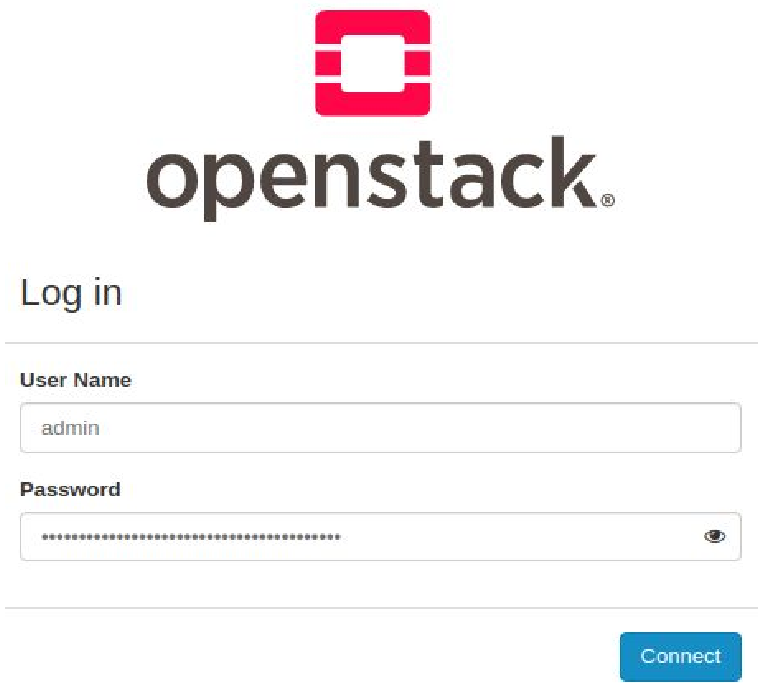
admin-openrc.sh is a file that consists of all data needed for authorization to keystone.
In my case:

```
export OS_AUTH_URL=https://192.168.1.212:5000/v3
export OS_PROJECT_ID=e825c98695e540adb855adb8a75e9543
export OS_PROJECT_NAME="admin"
export OS_USER_DOMAIN_NAME="Default"
if [ -z "$OS_USER_DOMAIN_NAME" ]; then unset OS_USER_DOMAIN_NAME;
fi
export OS_PROJECT_DOMAIN_ID="default"
if [ -z "$OS_PROJECT_DOMAIN_ID" ]; then unset
OS_PROJECT_DOMAIN_ID; fi
unset OS_TENANT_ID
unset OS_TENANT_NAME
export OS_USERNAME="admin"
echo "Please enter your OpenStack Password for project
$OS_PROJECT_NAME as user $OS_USERNAME: "
read -sr OS_PASSWORD_INPUT
export OS_PASSWORD=$OS_PASSWORD_INPUT
export OS_REGION_NAME="RegionOne"
if [ -z "$OS_REGION_NAME" ]; then unset OS_REGION_NAME; fi
export OS_INTERFACE=public
export OS_IDENTITY_API_VERSION=3


# cd /usr/local/share/kolla-ansible/
# ./init-runonce
```

The last thing left is to find our password to openstack. To get it, run:

```
$ sudo grep keystone_admin_admin_password /etc/kolla/passwords.yml
```

You’d better save that one!



Glory Glory Hallelujah!

Part 8: Horizon Dashboard

In this part I will cover adding new networks, subnets, routers and instances.

To add new networks go to Network-> Networks or Network Topology and find a button named “Create Network”. Then, you have to add its name, subnet and assign a Network Address (This one is up to you, but remember about adding the mask in shortened format).

Adding instances is pretty similar, but you have to worry about many more things. (path: Compute-> Instances-> Launch Instance).

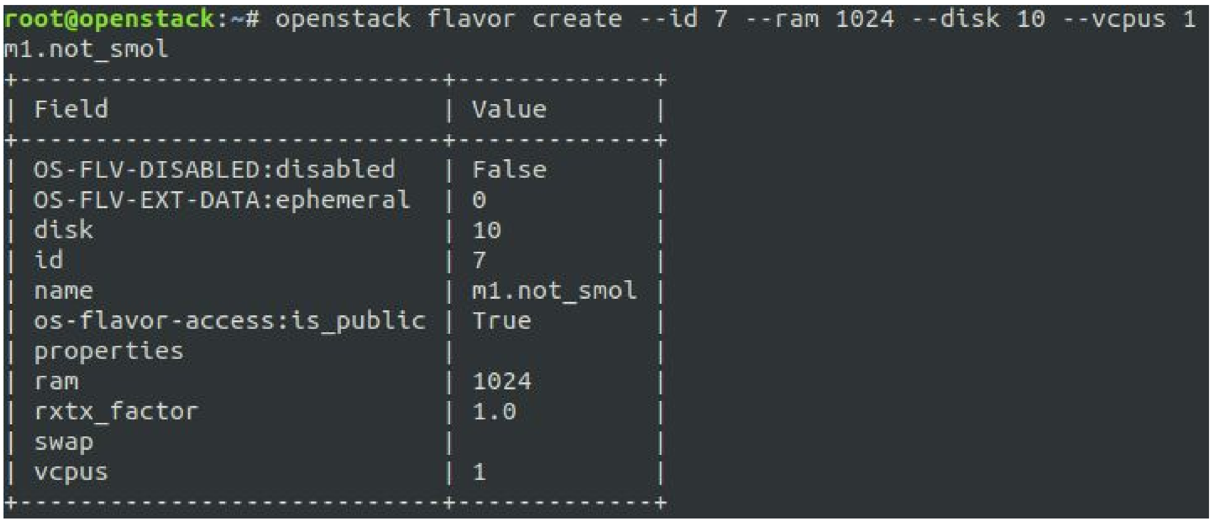
1. Add Instance name.
2. Boot from Image. By default you have only cirros. It is a simple lightweight OS for testing network functionalities.
   To set up a server, we have to add additional image. We can do it from CLI or GUI. Faster way is to use GUI.
   Now, you have to download CentOS or Ubuntu .qcow2. I chose to use CentOS 7 1905 from:
   https://cloud.centos.org/centos/7/images/
   Add it by entering the path Compute-> Images -> Create Image
   Select downloaded qemu2 and set minimum Disk as 9GB. Ready.
3. Flavor. As you can see, m1.tiny is too small to handle our newly added CentOS. Check m1.small.
4. Choose a desired network,
5. Then security group (we will configure it later) and

6. Key pair is used to allow your PC to ssh into that VM. To add your Key into Openstack platform follow these steps:
   On your PC run:
   ```
   $ ssh-keygen
   $ cat /your_home/.ssh/id_rsa.pub
   ```
   Copy the key and proceed to Compute -> Key Pairs -> Create Key Pair

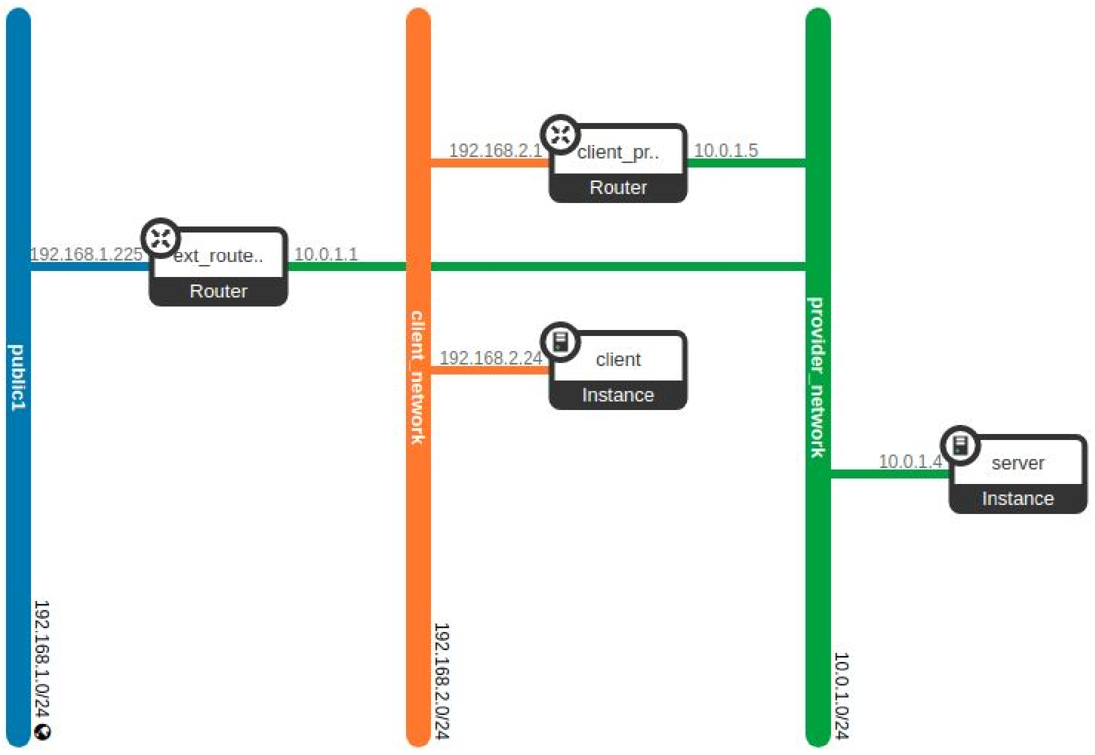
   In my case after starting an instance like this an error occurred. Referring to my hardware resources, I had not enough memory to launch that big instance (20GB). In order to deal with this error. I had to create new flavour.

7. Now we have to connect to our openstack and instance. To do so, it is necessary to edit security group. In order to do so, go to Project-> Network-> Security Groups
   It is not necessary to create new group, just edit the default one. To allow ping add this one rule:
   ```
   Egress     IPv4  ICMP  Any   0.0.0.0/0
   ```

Now it is possible to connect to openstack and instances through our PC. To add a flavour run this command:
```
$ openstack flavour create -- id 6 --ram 1024 --disk 10 --vcpus 1
m1.name
```
It will create a flavour that support 1GB RAM, 10 GB HD, 1 VCPU
In my case it looked like this:


```
root@openstack:~# openstack flavor create --id 7 --ram 1024 --disk 10 --vcpus 1
m1.not_smol
+----------------------------+-------------+
| Field                      | Value       |
+----------------------------+-------------+
| OS-FLV-DISABLED:disabled   | False       |
| OS-FLV-EXT-DATA:ephemeral  | 0           |
| disk                       | 10          |
| id                         | 7           |
| name                       | m1.not_smol |
| os-flavor-access:is_public | True        |
| properties                 |             |
| ram                        | 1024        |
| rxtx_factor                | 1.0         |
| swap                       |             |
| vcpus                      | 1           |
+----------------------------+-------------+
```

After all that configuration, we are able to set up a network that looks like this:



Part 9: Creating server

Now we have to ssh to our CentOS server instance. To set up a server, we have to install Apache, Nginx or similar application.

To install NginX:

```
$ sudo yum install epel-release
$ sudo yum install nginx
$ sudo systemctl start nginx
```

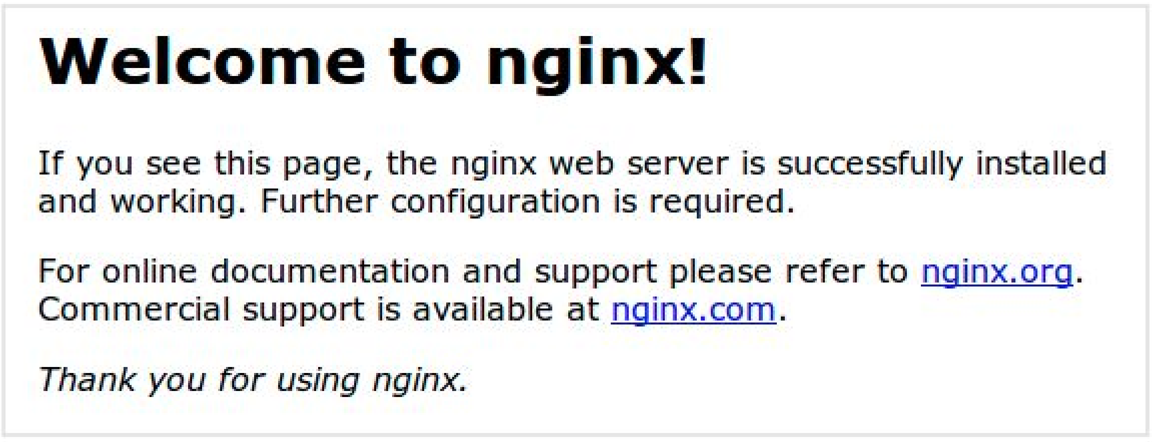
To allow our firewall to let the traffic come through, run:

```
$ sudo firewall-cmd --permanent --zone=public --add-service=http
$ sudo firewall-cmd --permanent --zone=public --add-service=https
$ sudo firewall-cmd --reload
```

To start NginX when booting instance run:

```
$ sudo systemctl enable nginx
```

Now, when everything is configured, we can go to the server’s floating IP address in browser. By default, we should see this page:


Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

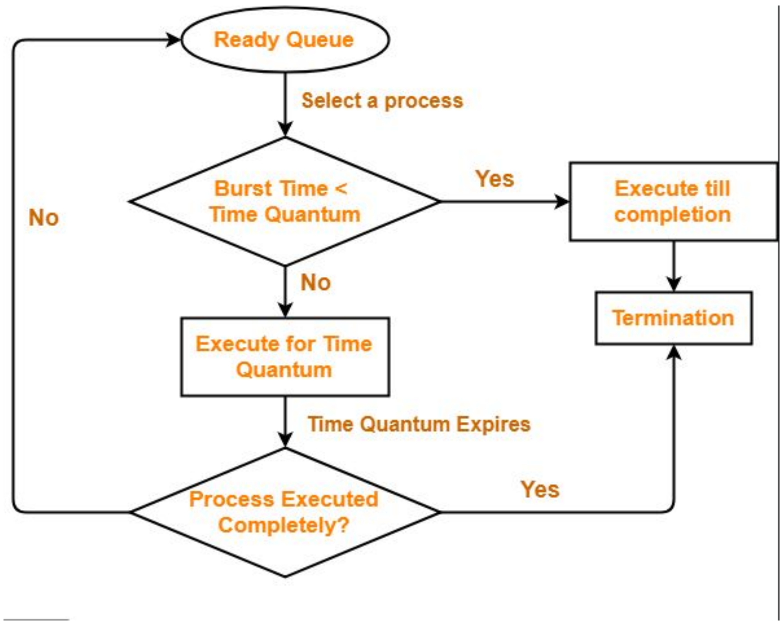
Thank you for using nginx.

To change load custom website, copy a custom file into `/usr/share/nginx/html.`
To configure NginX go to `/etc/nginx/nginx.conf.`

We also planned to support traffic unloading using Robin-Round algorithm but due to lack of resources, we couldn't create 2 servers to show it.

Round-robin - the simplest scheduling algorithm for processes in the operating system, which allocates appropriate time intervals to each process without any priorities. Therefore, all processes have the same priority.

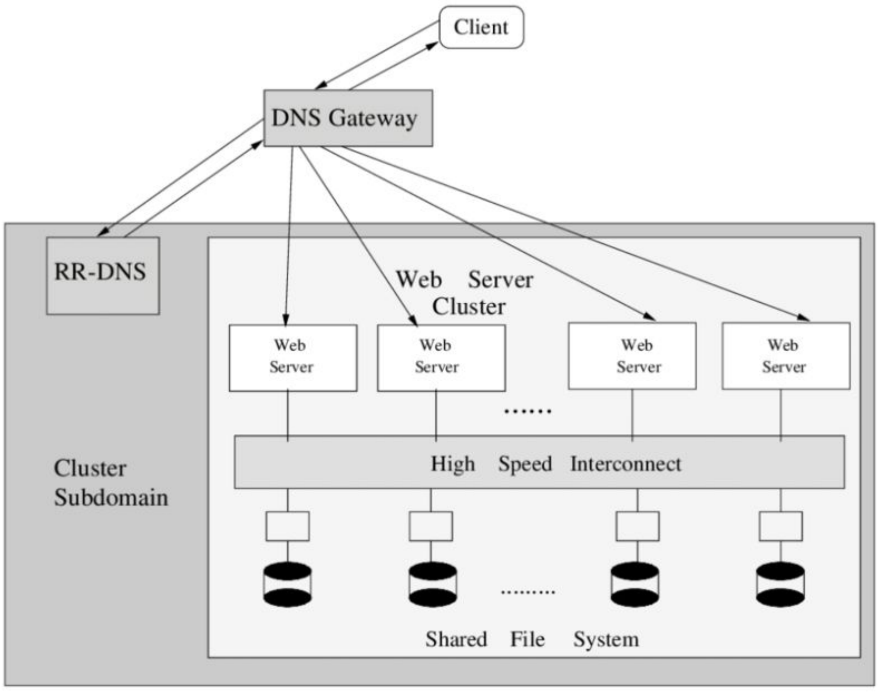
The diagram of Round-Robin algorithm



\

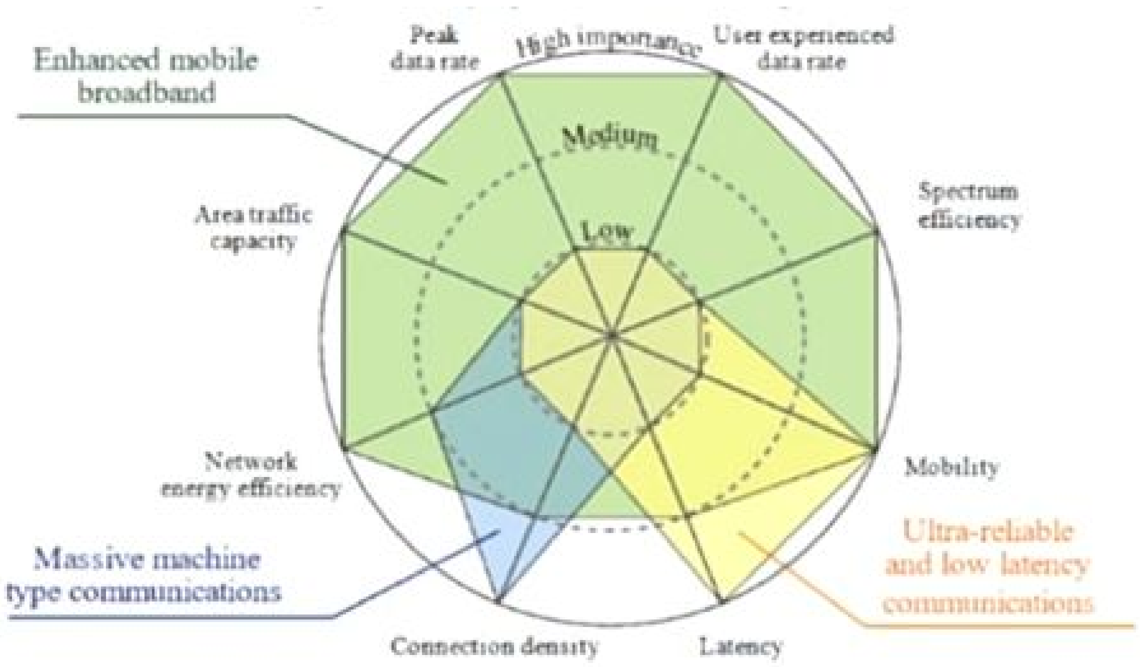
Round-Robin DNS

Multiple IP addresses are linked to a single Internet domain; it is the client who decides which server among the indicated ones wants to connect to. Unlike a load balancing system, this technique shows a partial internal structure and allows the client to access multiple servers.



Part 10: 5G slicing with OpenStack

With 5G networks showing up at the market there is one aspect of 5G architecture which become the hot topic. We are talking about 5G NS which means network slicing. With that feature operators will be able to customize networks for specific customers accordingly to their needs. As we know 5G will be better than 4G/LTE in every parameter. For every use case there is another set of optimized parameters.


Enhanced mobile broadband
Peak data rate
High importance
User experienced data rate
Medium
Low
Area traffic capacity
Spectrum efficiency
Network energy efficiency
Mobility
Massive machine type communications
Ultra-reliable and low latency communications
Connection density
Latency

With amount of devices which will be connected to 5G it's really important to adapt network to different needs of specific companies. Here comes OpenStack with help. To even do Network Slicing we will need NFV (Network Function Virtualization) and those virtualized network functions will be a part of a network slice. Also that might involve using some sort of SDN(Software-Defined Networking). Bushing through the internet I found ETSI GR NVF-EVE 012 diagram which gives an idea of where a network slicing engine might fit into.

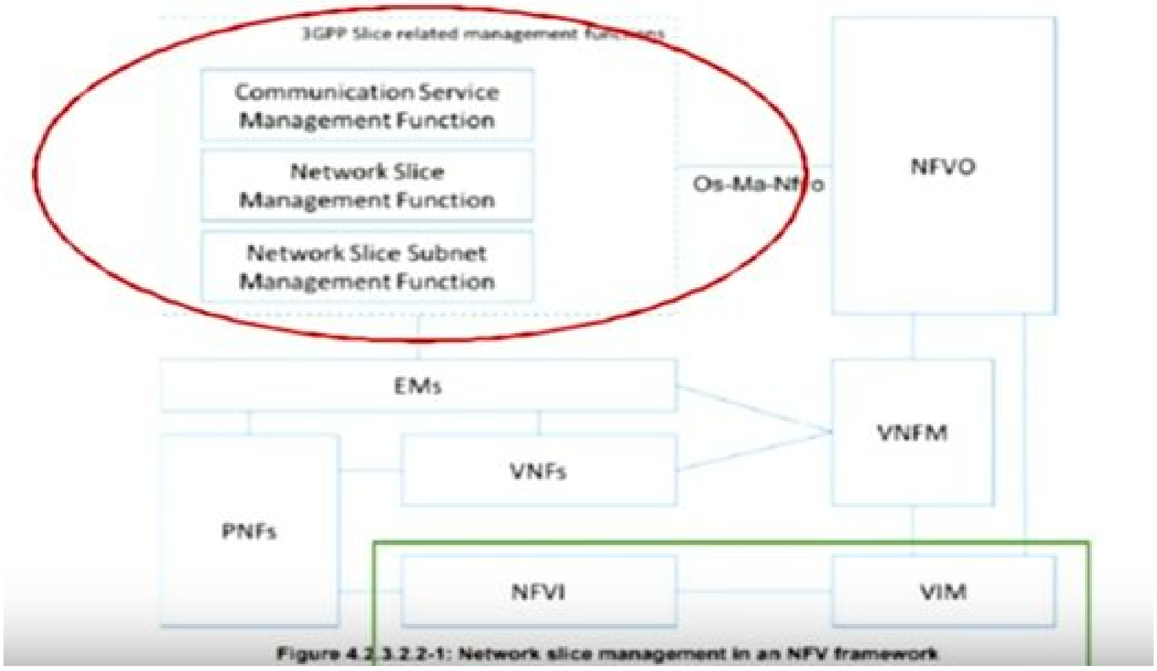

3GPP Slice related management functions
Communication Service Management Function
Network Slice Management Function
Network Slice Subnet Management Function
Os-Ma-Nfvo
NFVO
EMs
VNFM
VNFs
PNFs
NFVI
VIM

Figure 4.2.3.2.2-1: Network slice management in an NFV framework

So as we see in the diagram there is need to create some sort of functions to manage the network slices/ control their creation and communication services. To do network slicing we need Orchestrator – software responsible for automating the creation, monitoring and deployment of resources and services. An open source management and orchestration(OSM) stack has been developed in accordance with the ETSI NFV information models. Using the OSM orchestrator, an automated assurance and DevOps in service chains and 5G network slices are demonstrated on the picture below.

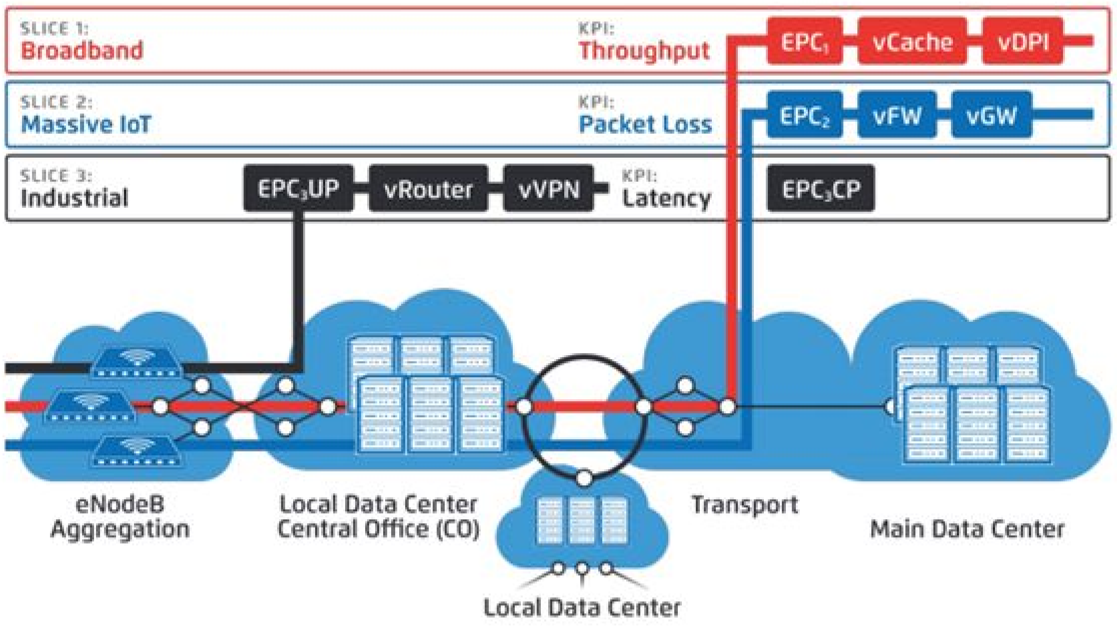


Figure 4. Three network slices with different critical key performance indicators[2]