Exercise 3

Welcome to exercise 3. This time we will have the following lessons:

1. Extend your knowlede about OpenStack: IaaS Networking
2. Multi-tier applications in Cloud Computing
3. Horizontal Scaling and Load Balancing

Lessons learned

- OpenStack provides virtual machines, security groups for firewalls, virtual networks and virtual routers.
- Wikimedia is a two-tier application with a presentation/logic tier and a data tier
- Horizontal scaling needs a load balancer as central point for user access. The data tier must not be part of the scaled presentation/logic tier.

Lesson 1: Extend your knowledge about OpenStack

OpenStack does not only provide virtual machines as shown in exercise 1 and 2. OpenStack has lots of projects, which extend the basic feature set.

Research: OpenStack Projects

Scan the official project navigator page at [1].

[1] https://www.openstack.org/software/project-navigator/

Question: OpenStack Projects

1. How is the project called, which provides virtual machines in OpenStack?

The compute service is responsible for virtual machines and is called nova. It cannot run alone: Keystone (Identity Service) and Glance (Image Service) are required at least.

2. What are the main differences between Object Store (Swift) and Block Storage (Cinder)?

Object Store is accessible via REST and allows applications to store arbitrary BLOBs. Cinder is accessible via iSCSI and provides volumes which can be attached to virtual machines as virtual disks.

Research: OpenStack Networking

OpenStack provides besides virtual machines virtual networks. The virtual network can be bridged to a public network or can be a private one. In the exercise so far, we used a public network to attach virtual machines directly to it. To hide some of your virtual machines from the public access, a private network is sometimes more useful. Such a virtual private network is then only accessible within attached virtual machines. To still have access to some of your virtual machines from the public, so called *floating IPs* can be used. Floating IPs can be assigned manually to virtual machines, while other IPs from public or private networks are assigned automatically by DHCP.

Question: Floating IPs vs. DHCP

What are the benefits and drawbacks of the manual assignment of floating IPs to virtual machines, compared to the automatic assignment in DHCP?

Task: OpenStack Networking

For the next steps in our exemplary mediawiki application, let's switch from plain public IPs to a virtual private network and public floating IPs.

Create a private network in OpenStack

In OpenStack, go to "Network -> Networks". Click "Create Network". Define the following settings:

- Network:
  - Network name: "private-net"
  - Admin State: "Up"
  - Create Subnet: Checked
- Subnet:
  - Subnet Name: "private-subnet"
  - Network Adress: "192.168.5.0/24"
  - IP Version: IPv4
  - Gateway IP:
  - Disable Gateway: unchecked
- Subnet Details:
  - Enable DHCP: Checked
  - Everything else:

Create a private router

The private network can now be used, virtual machines will get IP addresses automatically via DHCP. Yet, there is no connection to the public internet so far. Therefore we need a virtual router, which connects the private virtual network with the public network. Without this router, no public floating IPs can be assigned.

Go to "Network -> Routers". Click Create Router. Define the following settings:

- Name: "my-router"
- External Gateway: "routed"

Go to tab "Interface" of this router. Click "Add interface". Define the following settings:

- Subnet: "private-subnet"
- Everything else is default

From now on you should be able to create VMs in the network "private-net". These VMs will get a private IP address via DHCP. To access these VMs from the public internet, you can assign a floating IP to VMs which need public access. Not all VMs necessarily need to be accessible publicly (e.g. a database server). To still have a SSH connection you can do ssh hopping: log in to a VM with floating IP and from there tunnel through the VM you want to access. For more details, check the man pages of ssh [1] and search for ProxyCommand.

[1] https://linux.die.net/man/5/ssh_config

Move main_server to private network

We will now attach the existing virtual machine named main_server with the mediawiki application to the new private network.

- In the menu list of the main_server instance, select “attach interface”
- Select “private-net” and press “attach interface” button
- Verify, that your instance now has two interfaces (the existing one in network “public” and the new one in “private-net”)
- In the menu list of the main_server instance, select “detach interface”
- Select the address from the public network, and press the button to continue.
- Verify, that your instance now has only one interface with a private IP address (192.168.5.x)
- “Soft Reboot” the instance via the openstack dashboard to make sure it sets the new IP routing correctly.

The virtual machine is now not accessible any more from the outside. In lesson 3 we will create a new vm with a floating IP, from which you can tunnel through to your private network.

Lesson 2: Multi-tier applications in Cloud Computing

Before we start splitting the mediawiki application, it is necessary to have a clear understanding of multi-tier architectures.

Research: What are multi-tier applications?

Do some research on multi-tier architectures and applications. You could start with the Wikipedia page https://en.wikipedia.org/wiki/Multitier_architecture.

Finally, have a look at the general architecture of the mediawiki application at https://www.mediawiki.org/wiki/Manual:MediaWiki_a

Question: Multi-tier applications

1. Which tiers are typically used in a web-based application?
2. How many tiers has the mediawiki application?
3. Why is it useful to deploy each tier to a separate vm in cloud computing? Or why is it not useful?

Lesson 3: Horizontal Scaling and Load Balancing

In this lesson, we create a second vm with mediawiki, another vm with nginx as a loadbalancer, and we will extract the database into a separate vm.

Task: Create a second mediawiki vm

Create a new mediawiki vm, named "second_server" by using your snapshot from the previous exercise. This new VM must be attached to your private-net network! Make sure to use the same flavor as for main_server. Since the IP of the monitoring VM is unchanged, your new second_server should appear in the Chronograf dashboard automatically. Validate if the new vm shows up in Chronograf before you continue.

Task: Create the loadbalancer

1. Create a new VM named "loadbalancer" with flavor "m1.nano" and image "ubuntu 16.04". This new VM must be attached to your private-net network!
2. Add a Floating IP to the loadbalancer vm, by clicking "Associate floating ip". If "no floating IPs are allocated" is shown, click the "+" Button to allocate a floating IP from the "routed" pool.
3. Connect to the new loadbalancer vm via SSH

The loadbalancer vm is now accessible via floating ip from the public. But the vm is attached to the private network. OpenStack is taking care of the translation of floating ip to the ip of the vm in the private network. The vm itself is unaware of the floating ip (check the interfaces via `ip addr show`).

You can use this loadbalancer vm to access the two mediawiki servers via ssh hopping. You need to tunnel through the loadbalancer vm, e.g. via `ssh -o ProxyCommand='ssh ubuntu@FLOATING_IP nc PRIVATE_IP_TO_ACCESS 22' ubuntu@PRIVATE_IP_TO_ACCESS` or via ssh config and the ProxyCommand setting.

Lets install the loadbalancer VM with nginx (copied from the nginx website):

```
sudo -s
nginx=stable
add-apt-repository ppa:nginx/$nginx
apt-get update
apt-get install nginx
```

Now that the nginx is installed, we need to configure it:

1. create a new config file /etc/nginx/conf.d/wiki.conf for nginx and open it with your favourite editor (nano, vim, emacs, ...).
2. Copy and paste the following content into this file (replace the two IP addresses with the ones from your main_server and second_server):

```
upstream myproject {
  server YOUR_MAIN_SERVER_IP:80;
  server YOUR_SECOND_SERVER_IP:80;
}
server {
  listen 80;
  location / {
    proxy_pass http://myproject;
  }
}
```

Finally remove the default nginx site (rm /etc/nginx/sites-enabled/default) apply the configuration by restarting the nginx: `sudo service nginx restart` Validate that the load-balancer is working. Go to http://FLOATING_IP_OF_LOADBALANCER:80/ - you should see a apache default page.

Unfortunately, the mediawiki needs to be reconfigured, since the public IP of it is now the floating IP of the load balancer. Log in to both mediawiki vms and edit the file `/var/www/html/wiki/LocalSettings.php` and change the line `$wgServer = "http://134.60.x.y";` to fit your loadbalancer's floating ip. The mediawiki is now accessible through http://FLOATING_IP_OF_LOADBALANCER:80/wiki

Now that we have a load balancer and two mediawiki instances running, think about the locality of your data store. Do the following tests:

- Create a new page or edit an existing page. Store your changes
- Reload the wiki and look for your changes. Are they still there? Reload and check several times if needed.
- The changes are sometimes there, sometimes are missing. Why is this the case?

The random behaviour experienced is caused by the fact that both wiki instances store their data independently. We will have to replace the individual mariadb databases by a commonly used mariadb database on an own virtual machine.

Task: Extract database into separate vm

Set-up a new virtual machine named "database" from an ubuntu 14.04 image (the old one to be consistent with main_server). Configure key, network, and the like just as before. In addition, add a security rule that allows access to the mariadb port (3306). You may want to add a floating IP to access the new vm directly, or you can use the load balancer vm and ssh hopping.

Inside the new "database" vm, run the following commands:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install mariadb-server
```

Before we can export the database from your existing mediawiki vm and import it to the new one, don't forget to add the wikimedia user, database and permissions. For a step-by-step guide, please have a look at exercise 1. Since we will connect to the database from a remote host later on, add additionally the following permissions:

```
GRANT ALL PRIVILEGES ON wikidb.* TO 'wikiuser'@'%' IDENTIFIED BY 'password';
```

where % stands for any host.

Per default, mariadb is configured to listen only to connections from the local host. This has to be changed, as multiple wikimedia instances will access the database over the network. Use your favourite editor to open file /etc/mysql/my.cnf as sudo and change the bind address:

```
bind-address = '<database vm private ip>'
```

To apply the change in the configuration file, restart mariadb with `sudo service mysql restart`.

Ensure that it is possible to connect to the database over the network with your new user. From one of the mediawiki VMs, try the following:

```
mysql -u wikiuser -h <database vm private IP> -p
```

As the database is currently empty, we have to import the old database:

1. dump the database from *one* of the wiki VMs with:

   ```
   mysqldump -u wikiuser -p --databases wikidb > dump.sql
   ```

2. copy the dump.sql file as discussed in one of the earlier lessons, e.g. scp, sftp to the new database vm, e.g.

   ```
   scp -o ProxyCommand='ssh ubuntu@$PUBLIC_IP nc $PRIVATE_IP_MAIN_SERVER 22' \
     ubuntu@$PRIVATE_IP_MAIN_SERVER:~/dump.sql /tmp/dump.sql
   scp -o ProxyCommand='ssh ubuntu@$PUBLIC_IP nc $PRIVATE_IP_DB_SERVER 22' \
     /tmp/dump.sql ubuntu@$PRIVATE_IP_DB_SERVER:~/dump.sql
   ```

3. Import database on the new database vm with

   ```
   mysql -u wikiuser -p < dump.sql
   ```

Next, we have to tell both mediawiki instances to use the new database vm. For *both* vms, change the file `/var/www/html/wiki/LocalSettings.php` to fit the following configuration parameters:

```
$wgDBtype = "mysql";
$wgDBserver = <internal ip of database vm>;
$wgDBname = "wikidb";
$wgDBuser = "wikiuser";
$wgDBpassword = "password";
```

Task: Validate the scaled mediawiki application

Use the load balancer to access your mediawiki application. Make changes to some sites of your mediawiki to validate if the consistency is still a problem. Run the stress test from the last exercise 2.

As a bonus you can install telegraf on the new database vm and the new load balancer vm. Telegraf has also support for nginx, to get monitoring data from your load balancer.

Question: Stressing the horizontally scaled mediawiki

- How much requests per second have you achieved with the vertically scaled mediawiki setup?
- How much requests per second do you achieve now with the horizontally scaled mediawiki setup?
- Vertical scaling was quite limited to the maximum flavor size. Can you image the new bottleneck of horizontal scaling?