# Practice M6: Clustering and High Availability (Ubuntu)

This practice assumes that you are working in an on-premise environment

All tasks can be achieved under different configurations (host OS and/or virtualization solution) with the appropriate adjustments

This practice is oriented towards **Ubuntu 21.04**

## Part 1: Load Balancing

### Linux Virtual Server

For this part we will need an infrastructure like this one:

Chart

Description automatically generated

Machines can be with or without graphical environment

Network settings shown on the picture reflect the ones, used during the demonstration. You should adjust them according to your setup

Please note that **M1** should be set as a default gateway for **M2** and **M3**

#### Installation

Let’s log on to the first machine (**M1**) and install the prerequisites

**sudo apt update**

**sudo apt install ipvsadm**

#### Preparation

##### IP Forwarding (Method 1)

Enable IP forwarding

**echo 'net.ipv4.ip\_forward = 1' | sudo tee -a /etc/sysctl.conf**

Read the values from configuration files

**sudo sysctl -p**

Check if the value has been set

**sysctl net.ipv4.ip\_forward**

##### IP Forwarding (Method 2)

Alternatively, if the **ufw** is running, you must allow the traffic forwarding in its configuration

#### Configuration

Open the **/etc/default/ipvsadm** file for editing

**sudo vi /etc/default/ipvsadm**

Change **AUTO** to **true** to make the service start on boot

Change **DAEMON** to **master**

Set the **IFACE** to match your external interface name. For example, **ens160** or **eth0**

Save and close the file

Now, enable and start the service if not already started or restart it if already started

**sudo systemctl enable --now ipvsadm**

And check if everything is okay

**systemctl status ipvsadm**

Let’s check if there are any existing rules

**sudo ipvsadm -l**

We can achieve the same with

**cat /proc/net/ip\_vs**

Let’s clean the rules even if there aren’t any

**sudo ipvsadm -C**

Now, let’s add a virtual service that will listen on port **80/tcp** and will use the **round-robin** distribution method

**sudo ipvsadm -A -t <ext-ip-of-machine-1>:80 -s rr**

And now, we must add the two backend (or real) servers with

**sudo ipvsadm -a -t <ext-ip-of-machine-1>:80 -r <ip-of-machine-2>:80 -m**

**sudo ipvsadm -a -t <ext-ip-of-machine-1>:80 -r <ip-of-machine-3>:80 -m**

Let’s check what are the results so far

**sudo ipvsadm -l**

We should see the rules

It would be nice to open the necessary (**80/tcp**) firewall ports on **M1** if the firewall is running

#### Testing

In order to prove that our setup is working correctly, we must install a web server on both **M2** and **M3**

Install Apache web server

**sudo apt update**

**sudo apt install apache2**

Enable and start it if not already started

**sudo systemctl enable --now apache2**

Set a custom index page

**echo 'Hello from M2' | sudo tee /var/www/html/index.html**

It would be nice to open the necessary firewall ports (**80/tcp**) as well if the firewall is enabled and started

Make sure that the above steps were executed successfully on both stations

Open a browser tab on the host (or on the **M1** machine) and navigate to **<ext-ip-of-machine-1>**

Now refresh a few times to see that the page is served by different nodes

#### Persistent Configuration

In order to retain the configuration after reboot, we must save it

**sudo ipvsadm-save -n | sudo tee /etc/ipvsadm.rules**

Alternatively, we can use the following pair of commands to save and load the rules

**sudo ipvsadm -Sn > rules.txt**

**sudo ipvsadm -R < rules.txt**

### Linux Virtual Server + Keepalived

For this part, we will need an infrastructure like this

Chart, diagram, waterfall chart

Description automatically generated

#### Installation (Load Balancers)

Log on to **M1**

Install the required packages

**sudo apt update**

**sudo apt install keepalived ipvsadm**

Repeat the installation on **M2** as well

#### Installation (Clients)

Let’s prepare the other two machines as well

Log on to **M3** and execute

**sudo apt update**

**sudo apt install apache2**

The service should start automatically, but if not, then enable and start it

**sudo systemctl enable --now apache2**

Repeat this on **M4** as well

#### Configuration (Clients)

Set a custom index page

**echo 'Hello from WEB1' | sudo tee /var/www/html/index.html**

It would be nice to open the necessary firewall ports (**80/tcp**) as well if the firewall is running

**sudo iptables -t nat -A PREROUTING -d <ip-address-m3> -j REDIRECT**

**sudo iptables -t nat -A PREROUTING -d <vip-address> -j REDIRECT**

Repeat the above on **M4** as well

#### Configuration (Load Balancers)

Log on to **M1**

You can either create a copy of the provided sample configuration file

**sudo cp /usr/share/doc/keepalived/samples/keepalived.conf.sample /etc/keepalived/keepalived.conf**

And then adjust it according to the provided **LSAA-M6-P1-KA1.txt** file

Or use the provided **LSAA-M6-P1-KA1.txt** file and copy it as **/etc/keepalived/keepalived.conf**

No matter which way you will go, don’t forget to adjust the IP addresses, names, and other parameters you see fit and most especially the name of the interface - substitute **eth0** with the appropriate value

Save and close the file

Now, enable and start the **keepalived** service

**sudo systemctl enable --now keepalived**

Check if the virtual address is there

**ip a**

Adjust the following settings in **/etc/sysctl.conf**

**echo 'net.ipv4.ip\_forward = 1' | sudo tee -a /etc/sysctl.conf**

**echo 'net.ipv4.conf.all.rp\_filter = 0' | sudo tee -a /etc/sysctl.conf**

Apply the changes with

**sudo sysctl -p**

Open the **/etc/default/ipvsadm** file for editing

**sudo vi /etc/default/ipvsadm**

Change **AUTO** to **true** to make the service start on boot

Change **DAEMON** to **master**

Set the **IFACE** to match your external interface name. For example, **ens160** or **eth0**

Save and close the file

Check the status of the **ipvsadm** service and if not started, then do it

**sudo systemctl enable --now ipvsadm**

Check if the rules appeared with

**sudo ipvsadm -Ln**

Don’t forget to adjust the firewall as well if needed

Repeat the procedure on **M2** but change the **/etc/keepalived/keepalived.conf** to match the **LSAA-M6-P1-KA2.txt**

Adjust the configuration if needed

#### Testing

Open a browser tab and navigate to **<virtual-ip>**

Now refresh a few times to see that the page is served by different nodes

Hm, it seems that one and the same host replies

Open the **/etc/keepalived/keepalived.conf** file and comment out the **persistence\_timeout 50** setting (line 34)

Save and close

Change the configuration on both servers (**M1** and **M2**)

Restart the service (**keepalived**) on both servers

Check the generated rules

Refresh the browser tab several times

Now, the result is different

Now, you can stop one of the nodes and try to refresh again

### Load Balancing (HAProxy)

For this part, we will need an infrastructure like this

Waterfall chart

Description automatically generated

#### Preparation (Clients)

Let’s first prepare the two web servers

Log on to **M2** and install the software

**sudo apt update**

**sudo apt install apache2**

Set a custom index page

**echo 'Hello from WEB1' | sudo tee /var/www/html/index.html**

It would be nice to open the necessary firewall ports (**80/tcp**) as well if it is enabled and active

Open the main configuration file of **Apache**

**sudo vi /etc/apache2/apache2.conf**

Change (substitute **%h** with **\"%{X-Forwarded-For}i\"**) the log format setting (row 213) to

**LogFormat "\"%{X-Forwarded-For}i\" %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined**

Save and close the file

Restart the service

**sudo systemctl restart apache2**

Repeat this on **M3** as well

#### Preparation (Load Balancer)

Log on to **M1** and install the software

**sudo apt update**

**sudo apt install haproxy**

Let’s adjust the configuration file **/etc/haproxy/haproxy.cfg** to match our needs

**sudo vi /etc/haproxy/haproxy.cfg**

Go to the end of the file

Create a new **frontend** section like this

**frontend http-in**

**bind \*:80**

**default\_backend web\_servers**

**option forwardfor**

Create a new **backend** section like this

**backend web\_servers**

**balance roundrobin**

**server m2 <ip-address-m2>:80 check**

**server m3 <ip-address-m3>:80 check**

Be sure to adjust the parameters according to your setup

Save and close the file

We can configure the **rsyslog** daemon to capture the logs from **haproxy**

Open the **/etc/rsyslog.conf** file for editing

**sudo vi /etc/rsyslog.conf**

Uncomment the lines (17 and 18) about **imudp**

Just after them add this one

**$AllowedSender UDP, 127.0.0.1**

Save and close the file

Restart the **rsyslog** service

**sudo systemctl restart rsyslog**

Restart the **haproxy** service

**sudo systemctl restart haproxy**

Open the **HTTP** service in the firewall if it is active

#### Testing

Open a browser tab and navigate to **<load-balancer-ip>**

Refresh a few times

## Part 2: High Availability

For this part we will need an infrastructure like this one:

Chart

Description automatically generated with medium confidence

Machines can be with or without graphical environment

Network settings shown on the picture reflect the ones, used during the demonstration. You should adjust them according to your setup

### Preparation

Let’s first log on **M1** and install **Apache**

**sudo apt update**

**sudo apt install apache2**

Create a new file **/etc/apache2/sites-available/server-status.conf**

**sudo vi /etc/apache2/sites-available/server-status.conf**

With the following content

**<Location /server-status>**

**SetHandler server-status**

**Require local**

**</Location>**

Save and close the file

Enable the configuration with

**sudo a2ensite server-status.conf**

Then reload the **Apache2** service

**sudo systemctl reload apache2**

Create a sample **index.html** page

**echo 'Demo page served by WEB1' | sudo tee /var/www/html/index.html**

Adjust the firewall rules if needed

Repeat the steps on **M2** as well

### Installation

Log on to **M1** and install the required packages

We can start with the installation of the main packages

**sudo apt install pacemaker pcs**

Once installed, the service (**pcsd**) will be automatically enabled and started

Check if the **hacluster** user has been created

**grep hacluster /etc/passwd**

Set its password, for example to **Password1**

**sudo passwd hacluster**

Adjust the firewall rules if needed to allow **high-availability** set of services

Repeat the steps so far on the second (**M2**) node as well

### Configuration

Return on the first node (**M1**)

Remove the file

**sudo rm /etc/corosync/corosync.conf**

Authorize among the nodes

**sudo pcs host auth m1.lsaa.lab m2.lsaa.lab**

Once the authorization is successful, generate the cluster configuration

**sudo pcs cluster setup demo m1.lsaa.lab m2.lsaa.lab --force**

Start the cluster

**sudo pcs cluster start --all**

Enable cluster auto-start

**sudo pcs cluster enable --all**

Check the status of the cluster

**sudo pcs cluster status**

List all cluster properties

**sudo pcs property list --all**

Let’s set a property

**sudo pcs property set stonith-enabled=false**

Check membership status

**sudo pcs status corosync**

We can view the cluster configuration in raw format

**sudo pcs cluster cib**

We can check the configuration consistency of a running cluster with

**sudo crm\_verify -LV**

Multiple **V** letters can be added to increase the verbosity level

### Resources

Before we create any resources, let’s explore what types are there

Check resource standards

**sudo pcs resource standards**

Alternatively, for older installations the **crmsh** (or **crm**) tool can be used to interact with the cluster

Get list of the available resource providers

**sudo pcs resource providers**

Retrieve the list of all resource agents

**sudo pcs resource agents**

Narrow down the agents to a particular provider

**sudo pcs resource agents ocf:heartbeat**

Now, that we are more or less aware, let’s create the virtual IP address for the cluster

**sudo pcs resource create clha\_ip ocf:heartbeat:IPaddr2 ip=<virtual-ip-address> cidr\_netmask=24 op monitor interval=30s**

Now, we can define the highly available web site as additional resource

**sudo pcs resource create clha\_web ocf:heartbeat:apache configfile=/etc/apache2/apache2.conf statusurl="http://localhost/server-status" op monitor interval=1min**

And then check the status

**sudo pcs status**

We may see that both resources reside on different nodes

We can define a constraint to guarantee that both will be together

**sudo pcs constraint colocation add clha\_web with clha\_ip INFINITY**

If we ask once again for the status

**sudo pcs status**

We will see that both are on one and the same node

We can go even further. We can set an order for starting the resources

**sudo pcs constraint order clha\_ip then clha\_web**

We can explore just the constraints with

**sudo pcs constraint**

Or add the **--full** flag at the end to see their IDs as well

### Testing

We can open a browser tab and navigate to **<clha\_ip>** or the **<virtual-ip-address>** we used for its creation

Next, we can stop one of the nodes

First check where the resources are running

**sudo pcs status**

Now stop the node, for example **M1**

**sudo pcs cluster stop m1.lsaa.lab**

Log on to the other node and check the status

**sudo pcs status**

Re-open the browser tab and check if the site is still available

Start the stopped node again

**sudo pcs cluster start m1.lsaa.lab**

Check the status again

**sudo pcs status**

## Part 3: Failover Clusters and Storage

### LVM Shared Storage

We can either start from the beginning, or continue from the previous part

No matter which way we will go, we must have the following infrastructure

Waterfall chart

Description automatically generated with medium confidence

#### Storage Preparation

First, we must have an **iSCSI** target (**M1**)

Then, we must authenticate both nodes with the target (**M2** and **M3**)

#### Target Preparation

Log on to **M1**

Install the required package

**sudo apt update**

**sudo apt install targetcli-fb**

Create a folder to store the **iSCSI** disk files

**sudo mkdir /var/lib/iscsi-images**

Start the administration tool

**sudo targetcli**

Switch to the **fileio** backend

**cd backstores/fileio**

Create an **iSCSI** disk

**create D1 /var/lib/iscsi-images/D1.img 2G**

Switch to the **iscsi** functions

**cd /iscsi**

Define a new target

**create iqn.2021-09.lab.lsaa:m1.tgt1**

Enter the target

**cd iqn.2021-09.lab.lsaa:m1.tgt1/tpg1/luns**

Create a LUN using the disk created earlier

**create /backstores/fileio/D1**

Adjust the access to the resource

**cd ../acls**

Register the initiators 1 (**M2**) and 2 (**M3**)

**create iqn.2021-09.lab.lsaa:m2.init1**

**create iqn.2021-09.lab.lsaa:m3.init1**

Enter the record (if not there already)

**cd iqn.2021-09.lab.lsaa:m2.init1/**

Set user and password

**set auth userid=demo**

**set auth password=demo**

Switch to the other one and set the credentials

**cd ../iqn.2021-09.lab.lsaa:m3.init1/**

**set auth userid=demo**

**set auth password=demo**

Exit the administrative tool

**exit**

Adjust the firewall if needed and active

Don’t forget to enable and start the following service in order the configuration to be loaded automatically on boot

**sudo systemctl enable --now rtslib-fb-targetctl.service**

#### Initiator Preparation

Log on to the **M2** machine

Install the initiator package

**sudo apt update**

**sudo apt install open-iscsi**

Open the initiator configuration file for editing

**sudo vi /etc/iscsi/initiatorname.iscsi**

Set the name to match to your situation, for example **iqn.2021-09.lab.lsaa:m2.init1**

Save and close the file

Adjust the authentication settings in **/etc/iscsi/iscsid.conf** file

**sudo vi /etc/iscsi/iscsid.conf**

Change the mode on line 46 to **automatic**

Uncomment **node.session.auth.authmethod** = CHAP (line 59)

Uncomment and adjust **node.session.auth.username** and **node.session.auth.password** (lines 70 and 71)

Save and close

Initiate a target discovery with

**sudo iscsiadm -m discovery -t sendtargets -p <m1-name>**

Confirm what we have discovered

**sudo iscsiadm -m node -o show**

Login to the target

**sudo iscsiadm -m node --login**

Confirm the established session

**sudo iscsiadm -m session -o show**

Repeat the procedure on **M3** as well

#### Cluster Preparation

Next step is to spin up the cluster on nodes **M2** and **M3**

Remember to use the **FQDNs** (for example, **m2.lsaa.lab** instead of **m2**) where applicable

Refer to sections ***Installation*** and***Configuration*** in***Part 2: Failover Clusters***

#### Shared LVM

Log on to machine **M2**

Open the file **/etc/lvm/lvm.conf** for editing

**sudo vi /etc/lvm/lvm.conf**

And change the **system\_id\_source** (row 1227) to

**system\_id\_source = "uname"**

Close and save the file

Check that the **LVM** system ID matches the node name returned by **uname**

**sudo lvm systemid**

**uname -n**

Repeat the procedure on **M3** as well

Return on **M2**

Create (or re-create) the partition with

**sudo parted -s /dev/sdb -- mklabel msdos mkpart primary 16384s -0m set 1 lvm on**

Initialize it as physical volume

**sudo pvcreate /dev/sdb1**

Create a volume group

**sudo vgcreate vg\_ha /dev/sdb1**

Check if the system id is correctly applied

**sudo vgs -o+systemid**

Create a logical volume

**sudo lvcreate -l 100%FREE -n lv\_ha vg\_ha**

We can check the result with

**sudo lvs**

Create a filesystem

**sudo mkfs.ext4 /dev/vg\_ha/lv\_ha**

#### Turn Off Automounting

We must make sure that the volume groups that will be managed by Pacemaker won’t be auto-loaded by the system

Let’s check currently known volume groups on **M2**

**sudo vgs --noheadings -o vg\_name**

Depending on our configuration we may see one (we just created) or more

Open again the LVM configuration file

**sudo vi /etc/lvm/lvm.conf**

Go to row 1408 and paste the following

**auto\_activation\_volume\_list = []**

Please note that if on your system there are other (for example system) volume groups, their names should be included in this list. Just the one managed by **Pacemaker** should be absent

Save and close the file

Rebuild the **initramfs** by executing

**sudo update-initramfs -u**

And reboot the node

Repeat the procedure on **M3** as well

#### Resources and Resource Groups

Return on **M2** to finalize the setup

Check that everything with the cluster is okay

**sudo pcs cluster status**

Create the mount point on all nodes (**M2** and **M3** in our case)

**sudo mkdir /shared-lvm**

Being on **M2**, create a new cluster resource

**sudo pcs resource create lvm\_ha ocf:heartbeat:LVM-activate vgname=vg\_ha vg\_access\_mode=system\_id --group ha\_group**

We can check the status

**sudo pcs status**

Or just the resources

**sudo pcs resource status**

Next, we must create the file system resource with

**sudo pcs resource create lvm\_fs ocf:heartbeat:Filesystem device=/dev/vg\_ha/lv\_ha directory=/shared-lvm fstype=ext4 --group ha\_group**

Let’s check again the cluster and the resources and if the mount was successful

**sudo pcs status**

**sudo pcs resource status**

**df -hT**

Create a simple text file there with

**echo 'Hello from Shared LVM' | sudo tee /shared-lvm/readme.txt**

#### Failover Test

We can stop first node **M2**

**sudo pcs node standby m2.lsaa.lab**

And then on the second node **M3** we can check if the resource is there and working

**sudo pcs status**

**lsblk**

**cat /shared-lvm/readme.txt**

Now, while on the second node bring back the first node

**sudo pcs node unstandby m2.lsaa.lab**

Check the status again

**sudo pcs status**

And move back the resource (if you want) to it with

**sudo pcs resource move lvm\_ha m2.lsaa.lab**

### NFS Cluster Resource

Let’s build on the previous set of tasks by adding NFS capabilities to our cluster while utilizing the shared **LVM**

Chart, waterfall chart

Description automatically generated

We can make a use of one additional station though – a client

#### Configuration

Log on to the first node **M2**

Install the **NFS** related packages

**sudo apt install nfs-kernel-server**

Adjust the firewall for **NFS** if needed

**sudo ufw allow nfs**

Repeat the above (package installation + firewall) steps on **M3** as well

Return on the node that runs the resources (assuming its node **M2**) and create required folders

The base folder we already have - **/shared-lvm**

We must create one more, for the actual exported folder

**sudo mkdir -p /shared-lvm/exports/share1**

Then, we must create the **nfsserver** resource

**sudo pcs resource create nfs\_daemon ocf:heartbeat:nfsserver nfs\_no\_notify=true --group ha\_group**

*You must adjust the names (resource, group, path) to match your settings*

Add the root **exportfs** resources to export the **/shared-lvm/exports** directory

**sudo pcs resource create nfs\_root ocf:heartbeat:exportfs clientspec=<network-address>/255.255.255.0 options=rw,sync,no\_root\_squash directory=/shared-lvm/exports fsid=0 --group ha\_group**

*You must adjust the names (resource, group, path) and the address to match your settings*

Add the actual share (export)

**sudo pcs resource create nfs\_share1 ocf:heartbeat:exportfs clientspec=<network-address>/255.255.255.0 options=rw,sync,no\_root\_squash directory=/shared-lvm/exports/share1 fsid=1 --group ha\_group**

*You must adjust the names (resource, group, path) and the address to match your settings*

Add the **floating IP** (or virtual IP) address resource that **NFS** clients will use to access the **NFS** share

**sudo pcs resource create nfs\_vip ocf:heartbeat:IPaddr2 ip=<virtual-ip-address> cidr\_netmask=24 --group ha\_group**

*You must adjust the names (resource, group) and the address to match your settings*

Check the status

**sudo pcs status**

While still on the active node, check the **NFS** exports with

**sudo showmount -e**

#### Testing

Log on to the client (**M4**)

Install the required package

**sudo apt update**

**sudo apt install nfs-common**

Check if the exports can be seen there

**sudo showmount -e <virtual-ip-address>**

Try to mount using **NFSv3**

**sudo mount -o "vers=3" <virtual-ip-address>:/shared-lvm/exports/share1 /mnt**

Check information about the mounted filesystem

**df -hT /mnt**

And unmount it

**sudo umount /mnt**

Try to mount the export but this time using **NFSv4**

**sudo mount -t nfs4 <virtual-ip-address>:share1 /mnt**

Check information about the mounted filesystem

**df -hT /mnt**

#### Failover Test

Return on the active node (for example **M2**)

Check the status of the cluster

**sudo pcs status**

And put the active node to standby

**sudo pcs node standby m2.lsaa.lab**

Go to the other (now the active one) node and check the status again

**sudo pcs status**

Go to the client machine and check if the mounted NFS export is working

Return on the second node and bring back the first node

**sudo pcs node unstandby m2.lsaa.lab**

Check the status again

**sudo pcs status**

And move back the resource (if you want or if not moved automatically) to it with

**sudo pcs resource move lvm\_ha m2.lsaa.lab**