**Implementation of auto-scaling**

The algorithm was made by understanding each individual component in a very precise way as the same feature of all the applications is converted into PHP cURL readable format (section 6.1.5 of Thesis) to work in sync with scaling algorithm. The whole implementation of this project is divided into several sub section:

1. Launching first web server from OpenStack CLI.
2. Installing host monitoring tool HsFlow to be gathered data and analysed by controller.
3. Installing PHP and setting up apache2 on web server.
4. Configure Load balancing using Haproxy to work with web server.
5. Separate PHP script testing for gathering information from OpenStack, sFlow metric, launching a new server, making snapshot of existing web server, and terminating a launched server.
6. After analysing output from each individual implemented in PHP scaling algorithm.

**Installing OpenstackClient on Ubuntu by:**

apt install python-dev python-pip

Instead of installing all these clients, I installed the only API The was needed in this project from OpenStack client. By replacing the <project> name in this then pip install command using the OpenStack services API name.

# pip install python-<project>client

For example : installing NOVA services I used

# pip install python-novaclient

Check if installation is done by typing:

#nova

Authentication and utilizing optimus the OpenStack controller

1>Open a terminal and switch to root: sudo su

2>apt-get install openvpn

3>cd to directory where tenant authentication bash file openrc.sh is kept in my case:

cd /home/amrit/Downloads/nclvpn/aktiwa (for lappy)

4>route add -net 192.168.7.0/24 gw 192.168.2.253

6>check the connection by ping 192.168.7.65/optimus

7>login to GUI http://optimus/

8>source openrc.sh every time after rebooting as its needed to gain access to user-only CLI commands. Openrc.sh would look like this.

export OS\_TENANT\_NAME=username\_tenant

export OS\_USERNAME=username

export OS\_PASSWORD=username

export OS\_AUTH\_URL="[http://192.168.7.65:5000/v2.0](http://192.168.7.65:5000/v2.0/)

**Launching an Instance from Openstack CLI**

After OpenStack Client API was installed as shown in appendix, a bash file with tenant authorization credentials was stored as given.

|  |
| --- |
| export OS\_TENANT\_NAME=username\_tenant  export OS\_USERNAME=username  export OS\_PASSWORD=username  export OS\_AUTH\_URL="[http://192.168.7.65:5000/v2.0](http:///h) |

Then, after sourcing this bash file, the access to OpenStack services is granted such as nova, keystone and neutron. Using neutron and nova API tenant network is made which provides internal network access for instances to isolates this network from other tenants. The tenant owns this internal network as it only provides network access for instances within it.

Creating the network using neutron api:

*neutron net-create final-net*

After running this command the output looks like this;



By passing the following command, the tenant network gets assigned a desired subnet attached to it just like the external network does. By default, the subnet will use DHCP so the nodes can obtain IP addresses.

*neutron subnet-create final-net --name final-subnet \--gateway 30.30.1.1 30.30.1.0/24*

After this a router was created and attached to the subnet created in previous step, then the router was attached to the external network by setting it as the gateway by issuing following commands:

*$ neutron router-create aktiwa-router*

*$ neutron router-interface-add aktiwa-router final-subnet*

*$ neutron router-gateway-set aktiwa-router external-net*

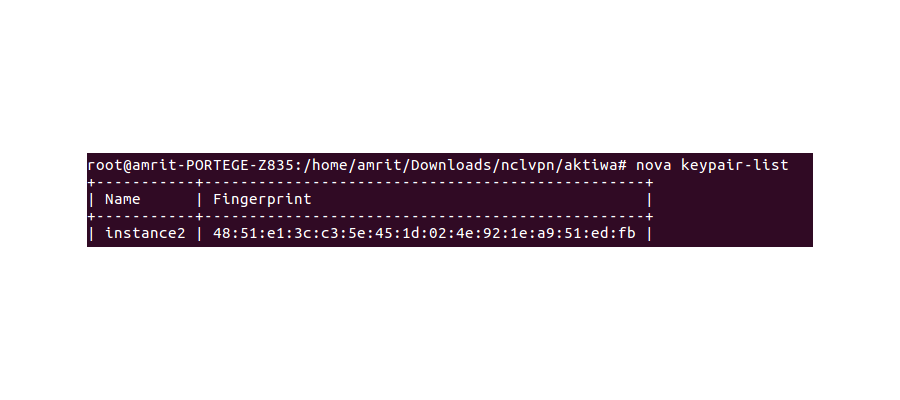
After creating tenant network a key pair was generated and public key was added to the OpenStack environment:

*$ ssh-keygen*

*$ nova keypair-add --pub-key ~/.ssh/id\_rsa.pub instance2-key*

Generated key pair can be viewed by

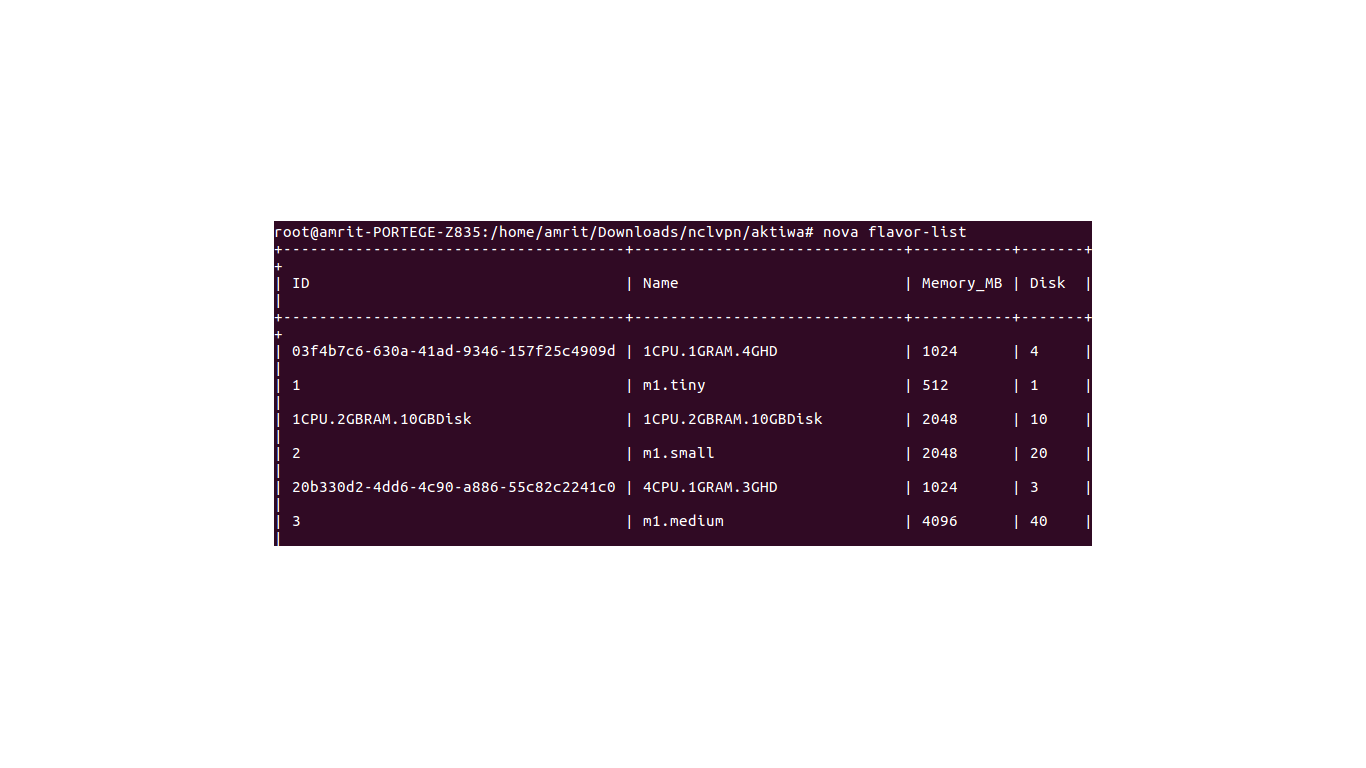
*nova keypair-list*



To launch a node

To launch an instance, the flavour, image name, network, security group, key, and instance name was specified.

1. A flavour specifies a virtual resource allocation: such as processor, storage and memory.
2. List avalable flavours: nova flavor-list



For the algorithm m1.small and m1.tiny are used as spawning these servers’ takes less time meaning more efficiency.

Similar process was repeated for assigning security group to default, and checking image list. After all these components was set the first web server (node3) was launched using the above resources

*nova boot --flavor m1.tiny --image Ubuntu 14.04.4 LTS (Trusty Tahr) --nic net-id=Final\_NET\_ID \  
 --security-group default --key-name instance2 node3*

After server was launched to be able to SSH it two important rules were added to the default security group:

Permit secure shell (SSH) and [ICMP](http://docs.openstack.org/icehouse/install-guide/install/apt/content/launch-instance-neutron.html) (ping):

*$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0  
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0*

To be able to SSH to any OpenStack instances, they need to be assigned a floating IP address which, by definition, is an IP address that a tenant project can associate with a node so that the instance has the same public IP address each time that it boots. Both, private IP address and floating IP address, are used at the same time on a single network.

The floating IP address is used for accessing the node/server from public domain and the private IP address is likely to be used for communication between instances between private networks. This node was connected via floating ip and the key pair generated before.

*ssh -i instance2.pem* [*ubuntu@192.168.7.177*](mailto:ubuntu@192.168.7.177)

**HSFlow Installation:**

1>sudo su

2>cd /etc

3>wget https://github.com/sflow/host-sflow/releases/download/v1.28.3/hsflowd-Ubuntu12\_1.28.3-1\_amd64.deb

4>unpack (optional install or vice versa)

**dpkg --unpack hsflowd-Ubuntu12\_1.28.3-1\_amd64.deb**

//optional//dpkg -i hsflowd-Ubuntu12\_1.28.3-1\_amd64.deb

5>after locating hsflowd.conf following line were added by

nano hsflowd.conf

Commented every line especially additional ‘DNSSD=on’

And these line within the parameter of sflow

DNSSD = off

polling = 1

sampling = 512

collector {

ip = 192.168.7.65

udpport = 6343

}

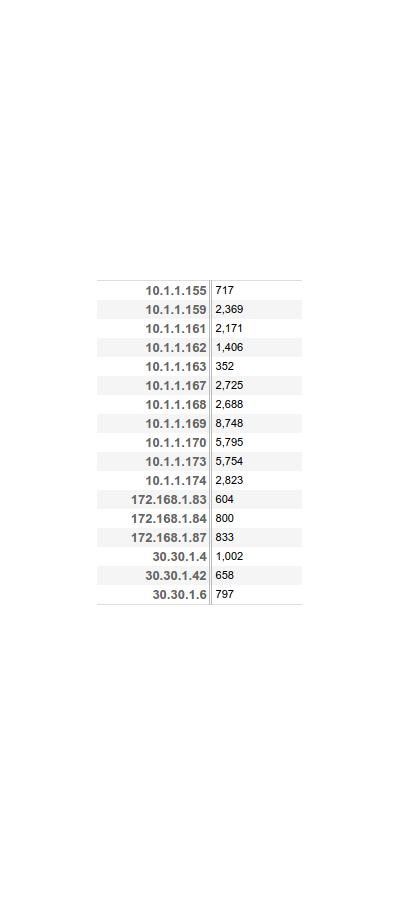
6>service hsflowd status

\* hsflowd is not running

7>service hsflowd start

8>restart node

9> after following the installation process as illustrated in Appendix A the servers’ metrics can be viewed from <http://optimus:8008/agents/html>



*Figure : node3 (30.30.1.6) shown under Agent list from* <http://optimus:8008/agents/html>

**Apache2 and PHP installation:**

Installation for apache2 and PHP was carried by below commands.

sudo apt-get update

sudo apt-get install apache2

sudo apt-get install php libapache2-mod-php php-mcrypt php-mysql

The UFW firewall of the server was adjusted to Allow Web Traffic by making sure that firewall allows HTTP and HTTPS traffic. The process for this was to make sure that UFW has an individual profile for Apache which is Apache Full and, by giving the following command, we can look at my Apache Full profile, it's show that it enables traffic to ports 80 and 443:

*sudo ufw app list*

Followed by:

*sudo ufw app info "Apache Full”*

Output for these commands can be seen in the screenshot of my terminal.



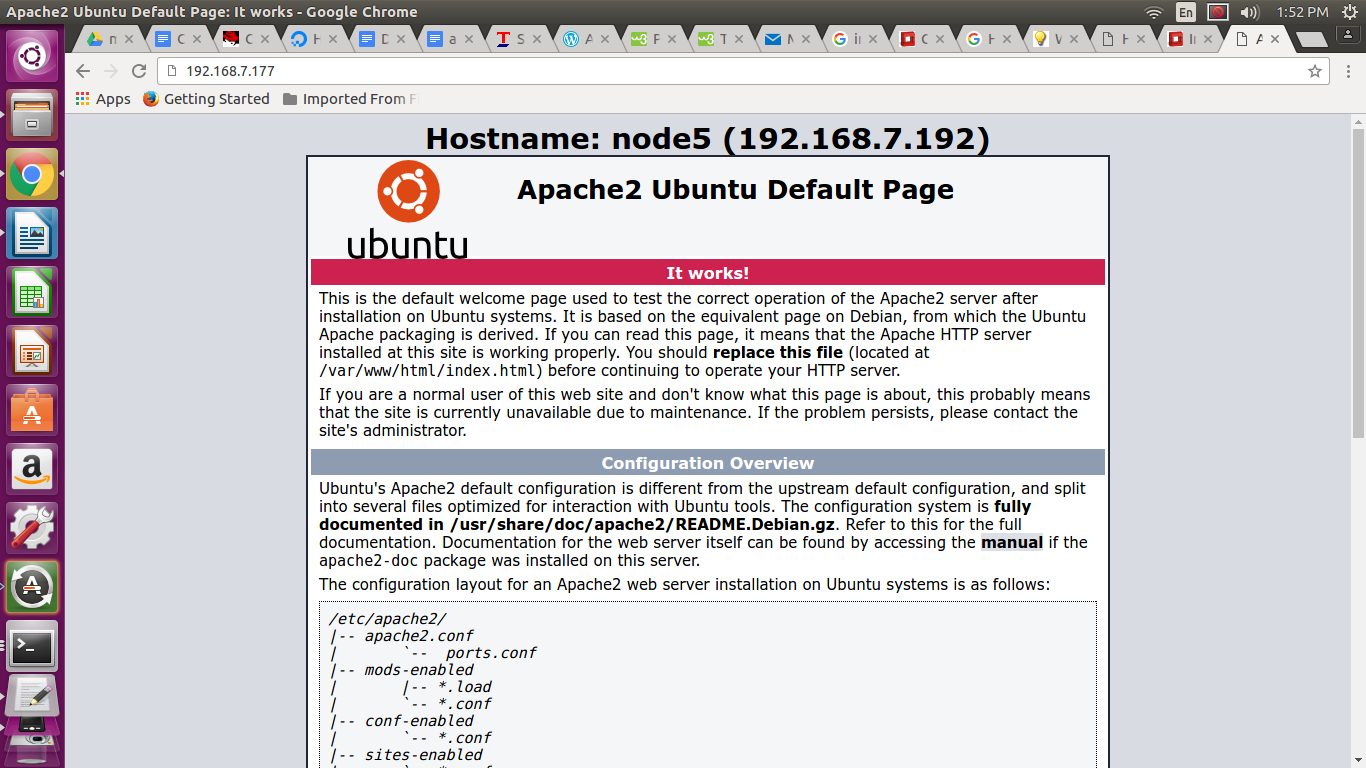
Once apache was configured, PHP was implemented as auto scaling algorithm is written in that. PHP is the component of this project setup that will help process and display code in dynamic content. Finally, hand over the processed content over to web server to display.

Once again by leveraging the apt-get system to install most of the project components I included some helper packages while installing PHP as shown in appendix, so that the algorithm run under the Apache server and if needed can talk to MySQL database

After both apache2 and PHP was installed by following simple commands, Apache serves files can be modified to change the way when the directory is requested. By default, Apache will first look for a file called index.html if a URL requests a directory from the server.

To verify, that everything worked perfectly after the above setup was completed, I did a spot check right away by visiting web server floating IP address in my web browser: http://192.168.7.177

This pops up the default informational and testing apache web page in Ubuntu 16.04. It looked something like this from web browser and if this page comes then its tells that web server is correctly installed and accessible through firewall:



*Figure : Apache was tested from different browser to check if HTTP or PHP based web page is accessible.*

**Installing HAProxy(Load balancer) using Ubuntu 14.04**

apt-get install haproxy

check version :

haproxy -v

Enable HAProxy to be started by the init script /etc/default/haproxy.

Set ENABLED option to 1 as:

ENABLED=1

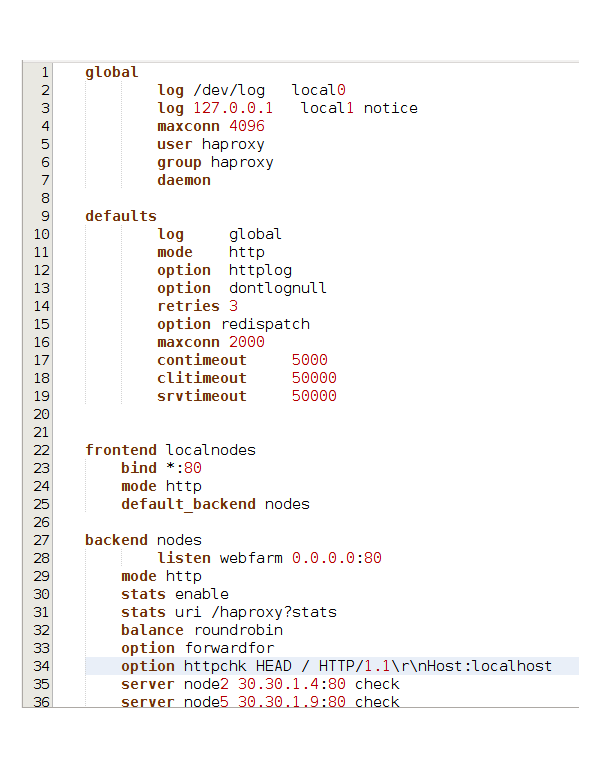
Verify change done properly by executing the init script of HAProxy without any parameters.

$ service haproxy <press\_tab\_key>

reload restart start status stop

Input these lines in /etc/haproxy/haproxy.cfg

sudo nano /etc/haproxy/haproxy.cfg



**Breakdown of haproxy.conf :**

global  
 log /dev/log local0  
 log 127.0.0.1 local1 notice  
 maxconn 4096  
 user haproxy  
 group haproxy  
 daemon

The directive *log:* Itmentions server to which log messages will be sent.

The directive *maxconn:*It specifies the number of concurrent connections on the front-end. The default value is *2000* and can be tuned according to machines/server configuration.

The directives *user* and *group:* These should not be changed as *it* changes the HAProxy process to the specified user/group.

defaults  
 log global  
 mode http  
 option httplog  
 option dontlognull  
 retries 3  
 option redispatch  
 maxconn 2000  
 contimeout 5000  
 clitimeout 50000  
 srvtimeout 50000

The section above has all the default values set autonomously when haproxy is installed.

*Redispatch:* Itenables redistribution of session in case of failure in connection is noticed. This way, session stickiness is overridden if any web server instance experience shutdown or system degradation..

Directive *retries: It* sets the number of connection retries on a web server instance after it is disconnected due to malfunction or connection failure.

Directives *timeout* needs to bemodified as well as *contimeout* option. It specifies the waiting period for an attempt to connect to a web server instance.

When the server node is expected to send data during the TCP process the options *clitimeout* and *srvtimeout* are applied. The recommended setting for HAProxy is to set timeouts in client and server to the same value.

listen webfarm 0.0.0.0:80  
 mode http  
 stats enable  
 stats url /haproxy?stats  
 balance roundrobin  
 option httpclose  
 option forwardfor  
 server node3 192.168.7.173:80 check //change this with the ip of backend server  
 server node4 192.168.7.177:80 check //change this with the ip of backend server

Above configuration shown is the main block which includes the settings for both the frontend(we tell HAProxy what to do with incoming requests.) and backend(mentioned nodes listens for incoming requests ). Here, HAProxy is configured to listen on port *80* for *webfarm,* which is the name given to help identify an application. When a request to HAProxy comes in on port 80, it will pass the request to one available server after checking the backend section for servers. The directives *stats* enable connection to statistics page. This page can be viewed with the URL mentioned in *stats url* which is *http://192.168.7.177/haproxy?stats* a screenshot of this page as viewed from my browser can be viewed.

**Siege :-**

On ubuntu

sudo apt-get install siege

From Terminal or CLI generate a config file

Siege.config

this commands generates a setting which is basically a hidden config file.

nano .siegerc

Once this is done siege lets create a stream of user specified in following command

siege -c 15 192.168.7.177

15 consecutive users will generate a performance test.

**For running Siege from a client add below lines in the two files mentioned**

**file1>in $HOME/.siegerc**

**connection = keep-alive**

**file2>in /etc/sysctl.conf**

**#delete below two lines if error http://stackoverflow.com/questions/19769436/siege-ports-getting-reused**

**net.ipv4.tcp\_tw\_reuse = 1**

**net.ipv4.tcp\_tw\_recycle = 1**

**#https://easyengine.io/tutorials/linux/increase-open-files-limit/**

**fs.file-max = 2097152**

**Run**

**sysctl -p**

**5 Auto Scaling Algorithm**

The auto scaling algorithm was designed after carefully implementing and analysing the above four crucial steps. After all the components were in the right place and able to communicate to each other, this scaling algorithm was implemented

the code for algorithm has been divided in several files which work together in harmony.

1. variable.php - Containing all the required variables and passing it to authentication file.
2. auth\_api.php - Opening line of communication between code and OpenStack Optimus.
3. snapshot\_creation.php - PHP script for creating snapshot of web server.
4. srv\_creation.php - PHP script for creating and launching server.
5. launching\_instance\_from\_snapshot.php - This is the main file which contains the scaling algorithm as well as includes rest of the files.

The file variable.php contains information such as OpenStack Authentication link "<http://192.168.7.65:5000/v2.0/tokens>" and Login Credentials such as tenant user id, admin id, and pass word of tenent. Apart from these, it provides OpenStack Services (compute, identity and networking) as seen from Keystone, Details of OpenStack Networking (external network and internal network), Details of image and flavour id, and Details of Admin sFlow Monitoring, which is basically the metric that will be monitored which for this project has been set to CPU Utilization.

The file auth\_api.php makes use of following API services: Keystone, Neutron, and Nova. Version 2 (V2.0) OpenStack APIs are mainly used in this project which is mentioned in the URL of each service API. A token is needed from OpenStack APIs to be generated for executing any service API served within OpenStack in this case Optimus. The token is only generated if there is a valid username and a password. A random key is generated as token combining both numbers and letters e.g. (e1h3v4j5n7k8l9m5v3v3v4j6c7x9b43f). The token has a very short availability time frame and either expires after and other token is generated in place of this. The token generated is included in X-Auth-Token header of all API requested.

The file launching\_instance\_from\_snapshot.php contains the dynamic scaling algorithm which make use of the back-end monitoring application sFlow-RT for this project which gathers data from Host sFlow application installed on every node. This combination of sFlow RT and HsFlow provides full visibility of machine various metrics such as CPU Utilization of the hypervisors also the hosted virtual nodes created on OpenStack Optimus. RESTful API is provided to Host sFlow that sends the collected data of all the resources to sFlow-RT which then is used in this file to determine when and how to scale these servers virtually.

The snapshot id of the newly created snapshot is passed to next file in line as an argument encapsulated in a variable. Later, file srv\_creation.php creates and launches a node similar to the node before by getting detail from snapshot id. The additional information which are: name of the server, image id to be used, flavor id needed, key pair name if it already exits , availability zone of the server, both internal and external network id, and security id. All the above detail is passed as an array which later using command json\_encode generates a string and return JSON representation of the object.

All these file is placed under default path /var/www/html/auto\_scale

The ip of server whose snapshot must be taken should be mentioned in file launching\_instance\_from\_snapshot.php

The value in variable file needs to be changed according to the ip of controller node.

The ip of backend servers should be mentioned in **haproxy.conf** in load balancer node.

Compile launching\_instance\_from\_snapshot.php

php launching\_instance\_from\_snapshot.php

The automation is supposed to work for a limited numbers of provided IP’s but as in the setup of optimus doesn't let you use the same ip even after it's released we have to add new ips to backend list of load balancer and replace previous ones.

1. The key file instance2.pem for SSH is stored in /home/aktiwa/Downloads
2. Change the backend server ip in

**ubuntu@node4:/etc/haproxy$ nano haproxy.cfg**

1. The snapshot will be created of server with 30.30.1.57 named instance1 to change the ip go to the launching\_instance\_from\_snapshot\_else.php make the change in the main section.

the load balancer can be checked if its working or not on the required nodes by visiting **http://load balancer ip /haproxy?stats**