Ighor Tavares

Table of Contents

1. Overview

For this remote challenge, I was given four of servers to work with:

IP-Address	Role	Service
54.184.205.242 (172.30.0.**200**)	Web Server A	Apache 2
35.167.192.239 (172.30.0.**161**)	Web Server B	Apache 2
18.236.198.70 (172.30.0.**109**)	Load Balancer	Haproxy
54.190.162.200 (172.30.0.**166**)	Server Monitoring	Nagios + Apache

- Access Nagios Control Panel:
  http://54.190.162.200/nagios/
  nagiosadmin
  nagios_expensify_2019

- Expensify user:

User:	expensify
Password:	expensify

- SSH Test User (After firewall)

User:	ighor
Password:	expensify19

2. Web Servers

Deciding on what webservice to use to host both web servers was easy. Apache seemed the ideal option since both servers are hosting a simple index.html. Apache is simple to set up and doesn't require any additional configuration in this case.

Before installing apache in both servers, I made sure to update the apt-get packages to ensure that the version of apache being installed is the latest stable one. I chose to run Apache as the webserver to host both Server A and B.

First, update the apt-get packages. Once I finished, I could proceed to install *apache2* in both servers.

```
sudo apt-get update
sudo apt-get install apache2
```

Once the installation was complete, I checked to ensure apache2 service was running properly with the following command: `sudo systemctl status apache2.` Since apache is now running in both servers, it is also a good idea to ensure that the service is being run by a non-root user. In this case, apache2 is running as *www-data* user.

Now I could proceed to create a simple index.html file in */var/www/html,* which displays a simple message telling the user which web server is currently being used. In this case, a simple <*h1*> tag with the message “ServerA” for one server and “ServerB” for the other.

The images below show both servers running after being set up.

3. Load Balancer

For the load balancer portion of the challenge, I decided to use *haproxy.* There are many other options, however I have previously worked with *haproxy* and I know that it provides a fast and reliable service, while being a completely free resource and having a lot of documentation. It was important to understand the requirements for the load balance so I could properly design the configuration. Installing *haproxy* is easy and straightforward:

First, we ensure that apt-get is up to date before installing *haproxy*

```
sudo apt-get update
sudo apt-get install haproxy
```

At first, since I wanted the load balance to stick with the web server and not change during repeated requests (change only if the current host is down and stick with the new web server), I thought the best approach would be a Layer 4 proxy. This is because it would only rely on one TCP connection, hence automatically sticking to the web server that our client connects to. However, keeping in mind that we are needed to send the client IP information to the web servers, having a Layer 4 proxy wouldn’t be possible without adding extra plugins which during my research I have found that most of them are deprecated. I chose to create a Layer 7 proxy instead, which focused on HTTP connections. The problem is, the Layer 7 configuration by default creates two TCP connections for each client connected: one with the load balance, and one with the web server connected to. In this case, the client would reconnect back to the top web server if it goes down and back up later.

In the configuration file, we first bind the ports that will be used by the load balancer to the current server IP address. We accomplish that in haproxy with the following lines in the file:

```
frontend localnodes
      bind *:60000-65000
      default_backend nodes
```

The next step was to configure the load balance to take any traffic from the range of ports given and redirect the traffic to both server's private IP through port 80. It is important to ensure that proper firewall rules are set for both web servers, which is demonstrated in the network management at the end of the report.

```
server serverA 172.30.0.200:80 check inter 5s fall 3 rise 99999999
server serverB 172.30.0.161:80 check backup
```

My first idea was to use one of the web servers as a main, and once it's down, it would use the other web server (server B) as the backup. However, it would fail to satisfy the requirements because once server A was back online and the client made the same request, they would be redirected back to server A instead of staying on server B.

A solution for this issue is to use server B as a backup if ServerA fails over. To avoid switching the current traffic back to ServerA, I added that it must do 99.999.999 healthy checks, with an interval of 5 seconds (which turns out to be approximately 15 years) in order to switch back to ServerA. While this is not optimal, it solved the issue. A better solution within *haproxy* would be to use sticky-tables to make the client "stick" to the destination server. However, I couldn't figure out how to make it work within my own configuration.

The full configuration file can be found at the end of this report.

4. Nagios Server Management

This was the first time I have ever worked with Nagios service. My first step was to understand and learn what Nagios is and how it works in the background. My primarily resource of study was Nagios docs page, although I spent a good amount of time learning from forums in Digital Ocean as well as from Nagios' own forum. There, I was able to learn how Nagios works and what services and ports it needs to function properly. In order to properly monitor different hosts other than the server running Nagios, I used the plugin NRPE (Nagios Remote Plugin Executor).

The monitoring requires to be hosted with a webserver and it needs to run PHP. So before installing Nagios in the server, apache and PHP need to be installed and configured.

First, update the apt-get packages and once finished, I could proceed to install apache2

```
sudo apt-get update
sudo apt-get install apache2
```

I could now proceed to install PHP. Additionally, helper packages were installed so it allows PHP to run under Apache web server.

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

Once apache and PHP are installed and the web server running, I could add the “Apache Full” application which enables traffic to ports 80 and 443 to the ufw firewall. I go more in detail about the firewall set up in the network management part of the report.

As a quick test to ensure apache is running and php is working, I created a simple php file that calls the `phpinfo()` function, so I can see the contents online. For best practice and security, this file was later removed from the server.

Now the server is ready to have Nagios installed. The first step is to create a user in the server that Nagios will use to run the service.

Create a user and group for Nagios and add this user to the group.

```
sudo adduser nagios
sudo groupadd nagcmd
sudo usermod -aG nagcmd nagios
```

Install development libraries, compilers, development headers and OpenSSL to properly build the Nagios Application

```
sudo apt-get update
sudo apt-get install build-essential libgd2-xpm-dev openssl libssl-dev
          unzip
```

Download the latest stable version of Nagios from their website

```
curl -L -O
https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.4.5.tar.gz
```

While reading about good practices while using Nagios, I learned that it is essential to configure the Nagios binary to ensure it is running as our recent created Nagios user and group. Before building the application, I ran the configure binary to set up group and command groups accordingly:

```
./configure --with-nagios-group=nagios --with-command-group=nagcmd
```

Once Nagios is properly configured, we can compile the application with `make all`. If compiled successfully, we can proceed to installing all the default configurations and init scripts with the following commands:

```
sudo make install
sudo make install-commandmode
sudo make install-init
sudo make install-config
```

Once Nagios was installed and configured, I was having issues connecting it to apache web server. In order to use apache to serve Nagios web interface, I first needed to copy the sample of the apache configuration file to the available sites folders: /etc/apache2/sites-available folder. I did that by installing the sample httpd.conf file to the target folder and named it for Nagios as shown below:

```
     sudo /usr/bin/install -c -m 644 sample-config/httpd.conf
/etc/apache2/sites-available/nagios.conf
```

In order to issue external commands via the web interface to Nagios, I had to add the webserver user www-data that runs apache to the group I created for Nagios, nagcmd**.**

```
Sudo usermod -G nagcmd www-data
```

The Nagios core application is installed and now I can proceed to install the NRPE plugin so it can collect data from different hosts. Nagios uses Nagios Remote Plugin Executor to monitor hosts in the network. It will use `check_nrpe` plugin in the main Nagios server and a daemon which it will run in the remote host to send data back to the Nagios server.

From the Nagios website, I can retrieve the latest stable release of the plugin. Then I proceed to download it:

```
curl -L -O
https://github.com/NagiosEnterprises/nrpe/releases/download/nrpe-3.2.1/nrpe-3.2.1.tar.gz
```

Once the download is finished and extracted, in the folder we first run the `./configure` to configure the check_nrpe plugin, and then install it with the following commands:

```
make check_nrpe
sudo make install-plugin
```

Once the plugin is done installing, I retrieved the information about what port Nagios will be running on. In this case it is port 5666.

Below is the configuration summary for the Nagios application:

*** Configuration summary for nrpe 3.2.1 2017-09-01 ***:

```
General Options:
-------------------------
NRPE port:              5666
NRPE user:              nagios
NRPE group:             nagios
Nagios user:            nagios
Nagios group:           nagios
```

Now that the plugin is installed, I can proceed to make a few configuration changes in Nagios as well as in apache in order to have everything running on the webserver and have the Nagios begin collecting data.

I learned that with Nagios, it must contain a configuration file for each remote host it plans to monitor. However, the configuration directory is disabled by default, but I can re-enable it and create my own path where the configuration files from the remote hosts can be stored.

First, I must edit the main Nagios configuration file:

```
/usr/local/nagios/etc/nagios.cfg
```

Now, I need to find and uncomment the following line in the file:

```
cfg_dir=/usr/local/nagios/etc/servers
```

Once done, I create the directory that will store the configuration file for each server that Nagios will monitor:

```
sudo mkdir /usr/local/nagios/etc/servers
```

In order to use check_nrpe command in the Nagios service, a new command needs to be added in Nagios configuration. Define a new command at the end of the file similar to the format as the other commands already present in it.

```
define command{

        command_name check_nrpe
        command_line $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
}
```

This defines a name and the command-line option to execute the plugin. Nagios will check the communication with the remote NRPE server (which it is monitoring).

Now I can enable Apache `rewrite` module which allows to rewrite requested URLs on the fly. I can also enable `cig` module that will allow Nagios to dynamically put content in the webserver.

For best practice, I created an admin user that can access the Nagios web interface. I accomplish that with the command `htpasswd`. For this challenge, I created the user `nagiosadmin` with the password `nagios_expensify_2019`, even though the password wouldn't be necessarily the safest to have. I also created a symbolic link for Nagios configuration file to enable its virtual host.

```
sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin

sudo ln -s /etc/apache2/sites-available/nagios.conf /etc/apache2/sites-
             enabled/  (symbolic link)
```

Restart Apache to load the new Apache configuration:

```
sudo systemctl restart apache2
```

The Nagios application, unfortunately, does not have a systemd unit file by default. It makes it more difficult to manage the service. It is possible to do this by creating a systemd unit file to manage the service in the `/etc/systemd/system/` directory. In this case, I called the file `nagios.service`. In the file, I added the following definition that specifies when Nagios should start and where `systemd` can find the service.

With the Apache configuration in place, I can set up the service for Nagios. Nagios does not provide a Systemd unit file to manage the service, so I will create one. Create the `nagios.service` file and open it in the editor:

```
/etc/systemd/system/nagios.service

[Unit]
Description=Nagios
BindTo=network.target

[Install]
WantedBy=multi-user.target

[Service]
Type=simple
User=nagios
Group=nagios
ExecStart=/usr/local/nagios/bin/nagios /usr/local/nagios/etc/nagios.cfg
```

Then, I must enable with systemctl and start the Nagios application:

```
sudo systemctl enable /etc/systemd/system/nagios.service

sudo systemctl start nagios
```

While testing Nagios, it was not able to execute commands, such as:

```
stderr: execvp(/usr/local/nagios/libexec/check_ping, ...)
```

My first approach was to check the folder where the function was trying to execute from. After some debugging and research, I learned that Nagios runs off plugins including the NRPE that I have installed so I can remotely execute plugins in the other servers. However, the core Nagios installation does not come with the essential plugins by default, like `check_ping`, `check_user` and so on. A quick solution that I learned is that Nagios contains an apt-get package `nagios-plugins`, that can be installed with `sudo apt-get install nagios-plugins`. Once the plugins are installed, I can copy all the plugins from the default folder to the folder where Nagios is executing the plugins from:

```
sudo cp plugins/check_* /usr/local/nagios/libexec/
```

Once completed, Nagios started reporting as it should.

Now that Nagios is up and running, we can proceed to install NRPE on the remote host computer. In this case, the other three servers that are running are both the webservers and the load balancer. As suggested by Nagios community and on their website, the NRPE plugin should run as separate user. Therefore, I created a Nagios user for all the monitored servers.

The first step is to create the Nagios user in all the servers which Nagios will monitor with `sudo useradd nagios`. The first time I did this, I downloaded the plugins needed for Nagios to execute commands. However, this time I learned in a forum that it is better to manually download from the website and properly set up the configuration for the plugins than depend on apt-get package. In order to build the plugin from scratch, I had to first install some dependencies:

```
sudo apt-get update
sudo apt-get install build-essential libgd2-xpm-dev openssl libssl-dev
              unzip
```

Once the dependencies are installed, the plugin can be configured to run as the user Nagios, built and installed.

```
cd ~
curl -L -O http://nagios-plugins.org/download/nagios-plugins-2.2.1.tar.gz
              tar zxf nagios-plugins-*.tar.gz
              cd nagios-plugins-*

./configure --with-nagios-user=nagios --with-nagios-group=nagios --with-
              openssl
make
sudo make install
```

In this case, the plugins are installed at the default directory where Nagios executes from, so we wouldn't have to manually move the executable plugin to a different directory as before.

Now that all the plugins are installed, I can proceed with NRPE plugin. It is a similar process to when I installed `check_npre` plugin in the main Nagios server, but here I added a few options for security such as SSL encryption during the configuration part.

I begin by downloading the nrpe plugin and configuring it to run as the Nagios user and tell it I want SSL support:

```
curl -L -O https://github.com/NagiosEnterprises/nrpe/releases/download/nrpe-
              3.2.1/nrpe-3.2.1.tar.gz

./configure --enable-command-args --with-nagios-user=nagios --with-nagios-
              group=nagios --with-ssl=/usr/bin/openssl --with-ssl-
              lib=/usr/lib/x86_64-linux-gnu

make all
sudo make install
sudo make install-config
sudo make install-init
```

At this point, I tested to check if the remote hosts were reporting back to the Nagios server, but I was missing a couple steps. I needed to ensure that in the remote hosts, they were to accept connection from the Nagios server. In the Nagios server, I needed to create a “monitoring” configuration file for each remote server Nagios is monitoring.

First in the remote hosts, I had to edit the `/usr/local/nagios/etc/nrpe.cfg` file , and add the Nagios server’s private IP to `allowed_hosts` line.

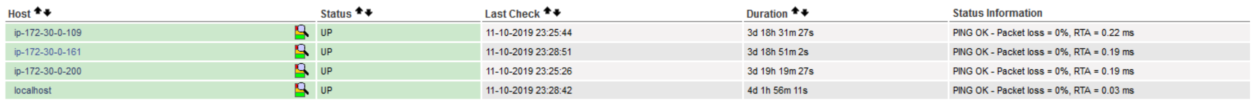
Then, in Nagios server, I created a new configuration file in `/usr/local/nagios/etc/servers/` for each of the remote hosts to be monitored. In the file, a new host needs to be defined and told what service we want to monitor. For this example, I used `ServerA` configuration file:

```
define host {
             use                                linux-server
             host_name                          IP OF SERVER A
             alias                              ServerA
             address                            PRIVATE IP OF SERVER A
             max_check_attempts                 5
             check_period                       24x7
             notification_interval              30
             notification_period                24x7
}

define service {
             use                                generic-service
             host_name                          IP OF SERVER A
             service_description                CPU load
             check_command                      check_nrpe!check_load
}
```

The service defined is to monitor the CPU load of the monitored remote host.

Once all the servers are properly set up, I can run `sudo systemctl restart nagios` to put the changes into effect and check on the web portal by logging in as the `nagiosadmin` account that was previously created and see all the remote hosts running.



Host	Status	Last Check	Duration	Status Information
ip-172-30-0-109	UP	11-10-2019 23:25:44	3d 18h 31m 27s	PING OK - Packet loss = 0%, RTA = 0.22 ms
ip-172-30-0-161	UP	11-10-2019 23:28:51	3d 18h 51m 2s	PING OK - Packet loss = 0%, RTA = 0.19 ms
ip-172-30-0-200	UP	11-10-2019 23:25:26	3d 19h 19m 27s	PING OK - Packet loss = 0%, RTA = 0.19 ms
localhost	UP	11-10-2019 23:28:42	4d 1h 56m 11s	PING OK - Packet loss = 0%, RTA = 0.03 ms

Because the process of adding NRPE plugin and creating the configuration file for each remote host was the same, I created two scripts that would accelerate the process. This way, there would be no need to manually repeat all the steps. I did not end up using it in the servers of this challenge, but the script was tested in a testing environment to ensure its functionality.

5. User Management

As requested, the user expensify was added to all the boxes with sudo privileges and with a default password “expensify” if needed. The next step was to properly setup the authentication through public key. While logged in as the expensify user, two things were accomplished:

Created .ssh folder with proper permissions
```
            mkdir ~/.ssh && chmod 700 ~/.ssh
```
Created authorized_keys file with proper permissions
```
            touch ~/.ssh/authorized_keys && chmod 600 ~/.ssh/authorized_keys
```

Once the files are properly set up, I can proceed to add the public keys provided and successfully connect via ssh with pair-key authentication.

6. Network Management

There had to be careful thinking while designing the network management of the servers.

Primarily, I had to consider how the traffic would transverse in the load balance to the web servers. I had to keep in mind that no one should be able to directly access the web servers - only through the load balancer. To accomplish that, I must only allow traffic through the port 80 from the load balancer.

I chose to use UFW for the firewall policies because it is easier to use and manage compared to ip-tables that can be messy to write and easier to make mistakes.

For the initial setup for both of the web servers (ServerA and ServerB), I added the following policies to accept traffic only from the load balancer.

```
            ufw allow from 172.30.0.109  to any port 80
            ufw allow from 18.236.198.70 to any port 80
            ufw deny http // deny every other traffic from port 80
```

In case of the load balancer, I must only allow traffic through ports 60000-65000 for incoming HTTP traffic that will be redirected to the web servers. I accomplished that by setting up a rule that allows traffic through this range of ports in a TCP connection.

```
            sudo ufw allow 60000:65000/tcp
```

Nagios uses port 5666 to communicate between the main server and the remote hosts. For each server, I allow the communication through this port with:

```
             sudo ufw allow 5666/tcp.
```

In the case of allowing only one server to be open for public ssh connection, the other servers can only be connected via ssh from this public server. While setting up the firewall, it must be defined that the connection through the port 22 can only be allowed from a specific IP address.

Here, it would be the private IP of the server that can be connected publicly. For this challenge, I chose to use the server running Nagios, since it is the central server that is monitoring all the other servers. For the firewall configuration and the Nagios server, I can add to allow ssh connections as the rule. For the other servers, I must define to only allow connection to port 22 from a specific ip address

Nagios server: `sudo ufw allow ssh`

Other servers: `sudo allow from <NAGIOS PRIVATE IP> to any port 22`

This was tested by creating a user in Nagios server named ighor and have its public key added to the other servers to ensure it can only be connected from that server.

Here is the overview of the configuration of all server's firewall

ServerA

```
     To                         Action      From
     --                         ------      ----
[ 1] 5666/tcp                   ALLOW IN    Anywhere
[ 2] 80                         ALLOW IN    172.30.0.109
[ 3] 80                         ALLOW IN    18.236.198.70
[ 4] 80                         DENY IN     Anywhere
[ 5] 22                         ALLOW IN    172.30.0.166
[ 6] 5666/tcp (v6)              ALLOW IN    Anywhere (v6)
[ 7] 80 (v6)                    DENY IN     Anywhere (v6)
```

ServerB

```
     To                         Action      From
     --                         ------      ----
[ 1] 5666/tcp                   ALLOW IN    Anywhere
[ 2] 80                         ALLOW IN    172.30.0.109
[ 3] 80                         ALLOW IN    18.236.198.70
[ 4] 80                         DENY IN     Anywhere
[ 5] 22                         ALLOW IN    172.30.0.166
[ 6] 5666/tcp (v6)              ALLOW IN    Anywhere (v6)
[ 7] 80 (v6)                    DENY IN     Anywhere (v6)
```

Load Balancer

```
     To                         Action      From
     --                         ------      ----
[ 1] 5666/tcp                   ALLOW IN    Anywhere
[ 2] 60000:65000/tcp            ALLOW IN    Anywhere
[ 3] 22                         ALLOW IN    172.30.0.166
[ 4] 5666/tcp (v6)              ALLOW IN    Anywhere (v6)
[ 5] 60000:65000/tcp (v6)       ALLOW IN    Anywhere (v6)
```

Nagios Server

```
     To                         Action      From
     --                         ------      ----
[ 1] Apache Full                ALLOW IN    Anywhere
[ 2] 22                         ALLOW IN    Anywhere
[ 3] Apache Full (v6)           ALLOW IN    Anywhere (v6)
[ 4] 22 (v6)
```

7. Resources

- https://exchange.nagios.org/directory/Addons/Monitoring-Agents/NRPE--2D-Nagios-Remote-Plugin-Executor/details

- https://www.digitalocean.com/community/tutorials/how-to-install-nagios-4-and-monitor-your-servers-on-ubuntu-16-04

- https://www.digitalocean.com/community/tutorials/how-to-set-up-a-firewall-with-ufw-on-ubuntu-16-04

- https://linuxhint.com/ufw_list_rules/
- https://discourse.haproxy.org/
- http://cbonte.github.io/haproxy-dconv/1.5/configuration.html#4-option%20forwardfor
- https://support.nagios.com/forum/
- https://exchange.nagios.org/directory/Addons/Monitoring-Agents/NRPE--2D-Nagios-Remote-Plugin-Executor/details

Haproxy Configuration File

```
log /dev/log  local0
        log /dev/log  local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin
        stats timeout 30s
        user haproxy
        group haproxy
        daemon

        # Default SSL material locations
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private

        # Default ciphers to use on SSL-enabled listening sockets.
        # For more information, see ciphers(1SSL). This list is from:
        #  https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/
        ssl-default-bind-ciphers
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:ECDH+3DES:DH+3DES:RSA+A
ESGCM:RSA+AES:RSA+3DES:!aNULL:!MD5:!DSS
        ssl-default-bind-options no-sslv3


defaults
        log     global
        mode    http
        option  dontlognull
        option redispatch # enables sessions redistribution in case of connection
failures. Stickness overriden if VPS goes down.
        timeout connect 5000
        timeout client  50000
        timeout server  50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http


frontend localnodes
        bind *:60000-65000
        default_backend nodes

backend nodes
        mode http
        option httpclose
        balance roundrobin
        cookie mycookie insert indirect nocache
        option forwardfor #tells service client information
        server serverA 172.30.0.200:80 check inter 5s fall 3 rise 99999999
        server serverB 172.30.0.161:80 check backup
```

NRPE Host Plugin Script

```
#!/bin/bash
# Quick install and setup for NPRE on a remote host once main server is setup

#add nagios user in host
sudo useradd nagios

#update apt-get and install dependencies

sudo apt-get update
sudo apt-get install build-essential libgd2-xpm-dev openssl libssl-dev unzip
```

```
#install Nagios plugins on remote hosts

cd ~
curl -L -O http://nagios-plugins.org/download/nagios-plugins-2.2.1.tar.gz
tar zxf nagios-plugins-*.tar.gz
cd nagios-plugins-*

#configure nagios plugins to use as nagios user and group and have SSL support
./configure --with-nagios-user=nagios --with-nagios-group=nagios --with-openssl

#compile and install nagios-plugin
make
sudo make install

# Install NRPE
cd ~
curl -L -O https://github.com/NagiosEnterprises/nrpe/releases/download/nrpe-
3.2.1/nrpe-3.2.1.tar.gz
tar zxf nrpe-*.tar.gz
cd nrpe-*

# Configure NRPE to run as Nagios user and group, and have SSL support
./configure --enable-command-args --with-nagios-user=nagios --with-nagios-group=nagios
--with-ssl=/usr/bin/openssl --with-ssl-lib=/usr/lib/x86_64-linux-gnu

#build and install NRPE and its default scripts
make all
sudo make install
sudo make install-config
sudo make install-init

#update NRPE configuration file
echo "Nagios Server's Private IP : "
read nagios_server_ip

sed -i "/allowed_hosts=127.0.0.1,::1/c\allowed_hosts=127.0.0.1,::1,$nagios_server_ip"
"/usr/local/nagios/etc/nrpe.cfg"

#start nrpe serive and check status
sudo systemctl start nrpe.service
sudo systemctl status nrpe.serice

#add to firewall
sudo ufw allow 5666/tcp
```

Adding New Remote Host cfg File Script (Nagios Server)

```
#!/bin/bash

echo "Input monitored server host name: "
read mshn

#sudo touch ${mshn}.cfg

echo "Input Client Server Alias: "
read csalias

echo "Input Monitored Server Private Ip :  "
```

```
read mspi

sudo echo "
define host {
      use                              linux-server
      host_name                        $mshn
      alias                            $csalias
      address                          $mspi
      max_check_attempts               5
      check_period                     24x7
      notification_interval            30
      notification_period              24x7
}

define service {
      use                              generic-service
      host_name                        $mshn
      service_description              CPU load
      check_command                    check_nrpe!check_load
}

" > /usr/local/nagios/etc/servers/${mshn}.cfg
```