Installation serveurs agrégation connexion 2014

Mis en place en 2014.

Introduction

L'objectif de ce guide est la mise en place de serveurs permettant l'agrégation de connexions. Dans mon cas ce sera une connexion ADSL avec une connexion 4G. Elle suppose l'installation de **un serveur un chez moi et un sur le net**. Entre ces deux serveurs nous construirons deux liens (avec tunnel VPN) un via la 4G et un via l'ADSL.

Tout sera connecté au serveur chez moi qui fournira une connexion internet unifiée. Même combat côté internet, mon serveur installé servira de point de sortie unifié vers le net.

Concernant la connexion ADSL, j'utilise la box normale de mon FAI. Concernant la connexion 4G, je prends un forfait internet qui va bien avec la carte SIM only. J'utilise ensuite ce modem 4G. Ca n'est pas obligatoire, c'est un choix perso. Je suppose que le modem 4G fourni avec le forfait par certains opérateurs de mobile peut aller également.

Vocabulaire :

- Je parlerai de **serveur maison***(ou maison en abrégé)* pour celui chez moi qui fera offcie de routeur / gateway pour mon bonding
- Je parlerai de **serveur dédié** *(ou dédié en abrégé)* pour le serveur chez online.net qui me sert de sortie sur le réseau public internet (via le réseau online.net)

1. Installation du serveur dédié

1.1. Installation initiale du serveur

Comme depuis un moment maintenant, je suis chez online.net. J'ai donc choisi assez naturellement leur petit serveur dédié, le SG Gen 2 de la gamme perso.

La première étape est assez simple chez online, après aller sur l'interface d'admin, on choisit sa distribution et on lance l'installation.Pour ma part je choisis ubuntu 14.04. Après hésitations avec la dernière debian, je suis finalement resté sur mes habitudes, d'autant plus que la dernière ubuntu est plus récente.

Pour ce serveur, l'installation de base ne requiert pas réellement de configuration spécifique.

Seule l'étape de partition nécessite (pour ma part en tous ças mais cela peut se discuter) un changement sur la taille de la SWAP : comme mon serveur possède 2go de mémoire, je préfère allouer un swap de 2048Mo au lieu du 1024Mo par défaut. Pour le calcul de la taille du SWAP

beaucoup de son de cloche existent. Pour ma part je me base sur cette page de la doc ubuntu. Je pars sur la fourchette basse (1x Taille de la RAM) puisque l'usage prévu du serveur sera vraiment peu gourmand en mémoire.

TODO SCREENS

1.2. Préparation de la configuration

Pour cette recette, il vous faudra utiliser les **3 ingrédients** suivants :

- OpenVPN *(voir ce lien et la manpage contient pas mal d'infos également)*
- Le système de bridging *(voir ce lien)*
- Le module de bonding *(voir ce lien et la doc officielle)*

1.2.1. Installation d'OpenVPN

```
sudo apt-get install openvpn
```

Nous utiliserons OpenVPN pour monter **2 tunnels VPN** l'un avec notre première connexion internet (la 4G) pour moi et l'autre avec notre seconde connexion internet (l'ADSL pour moi).

1.2.2. Installation du système de bridging

```
sudo apt-get install bridge-utils
```

1.2.3. Activation du module de bonding

C'est un module déjà présent dans l'install et avec le kernel ubuntu 14.04. Il faut cependant l'activer.

```
sudo vi /etc/modules
```

Ensuite faire en sorte qu'il s'active automatiquement au démarrage

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.

lp
rtc
bonding
dummy
```

Si vous voulez charger le module directement sans attendre un redémarrage

```
sudo modprobe bonding
```

1.2.4. Activation de l'ip forwarding

On voudra utiliser notre serveur comme routeur donc il est nécessaire d'activer le module ip forwarding.

```
sudo nano /etc/sysctl.conf
```

Et activer cette ligne

```
net.ipv4.ip_forward=1
```

1.2.6. Activation du NAT

C'est le nat qui va permettre le routage final.

Une bonne idée est de s'appuyer sur cette méthode : https://help.ubuntu.com/community/Router

Ce qui va donner chez moi ce genre de chose:

```
nano nat.sh
```

```
#
# Configure here your interfaces
#
EXTIF="bond0"
INTIF="br0"

#
# Commands path
#
DEPMOD=/sbin/depmod
MODPROBE=/sbin/modprobe

#
# General information
#
echo "[INFO][NAT] Loading NAT configuration..."
echo "[INFO][NAT] External Interface:  $EXTIF"
echo "[INFO][NAT] Internal Interface:  $INTIF"

#
# Checking requested linux kernel modules
#
echo "[INFO][NAT] Verifying that all kernel modules are ok"
```

```
$DEPMOD -a
echo "[INFO][NAT] Loading module ip_tables..."
$MODPROBE ip_tables
echo "[INFO][NAT] Loading module nf_conntrack..."
$MODPROBE nf_conntrack
echo "[INFO][NAT] Loading module nf_conntrack_ftp..."
$MODPROBE nf_conntrack_ftp
echo "[INFO][NAT] Loading module nf_conntrack_irc..."
$MODPROBE nf_conntrack_irc
echo "[INFO][NAT] Loading module iptable_nat..."
$MODPROBE iptable_nat
echo "[INFO][NAT] Loading module nf_nat_ftp..."
$MODPROBE nf_nat_ftp

#
# Enabling ip forwarding
#
echo "[INFO][NAT] Enabling forwarding..."
echo "1" > /proc/sys/net/ipv4/ip_forward
echo "[INFO][NAT] Enabling DynamicAddr..."
echo "1" > /proc/sys/net/ipv4/ip_dynaddr


#
# Setting iptables rules
#
echo "[INFO][NAT] Clearing any existing rules and setting default policy..."
iptables-restore <<-EOF
*nat
-A POSTROUTING -o "$EXTIF" -j MASQUERADE
COMMIT
*filter
:INPUT ACCEPT [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A FORWARD -i "$EXTIF" -o "$INTIF" -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
-A FORWARD -i "$INTIF" -o "$EXTIF" -j ACCEPT
-A FORWARD -j LOG
COMMIT
EOF
echo "[INFO][NAT] NAT configuration loaded !"
```

After configuring the 2 variables, save the script below as nat.sh and make it executable by doing

```
chmod a+x nat.sh
```

Now, test the script by running as root

```
sudo sh nat.sh
```

Investigate the messages from the console output to see if any error happened. If everything looks fine, use another host in the internal network to test if it can access the external network (presumably the Internet). A quick way to test is pinging Google public DNS from the console.

```
ping -c 3 -W 10 8.8.8.8
```

If ping responds, make our new script bootable so we don't have to run the script every time we restart.

```
sudo mv nat.sh /etc/init.d/
```

See :
https://askubuntu.com/questions/765120/after-upgrade-to-16-04-lts-rc-local-not-executing-command

```
sudo systemctl enable rc-local.service
```

```
sudo nano /etc/rc.local
```

```
# Executing nat script
source /etc/init.d/nat.sh
```

As a final test, restart your computer and test to see if you still have the same functionality. If so then congratulations! If not then make sure you followed the above correctly so the script is bootable.

Mais pour mémoire le point important est celui-ci:

```
iptables -A POSTROUTING -t nat -j MASQUERADE
```

Pour ne pas perdre le NAT, lancer cette commande à chaque boot par exemple en l'incluant dans les fichiers de démarrage (déjà pris en compte dans le lien de référence ci dessus).

1.2.5. Récupérer des ip / mac online.net

Dans le montage choisi, j'utiliserai un montage vpn en mode bridge. Résumé très vulgairement, l'idée est de monter mon serveur online, mon serveur à la maison sur le réseau online avec des ip publiques. Le montage sous forme de bridge permet cela en reliant ces serveurs avec une sorte de switch virtuel tant avec le mode bridge d'OpenVPN que le système de bridge linux.

Evidemment, on ne peut pas aller sur le réseau d'un hébergeur avec l'ip de son choix, il faut donc passer par l'étape d'obtention d'ip failover.

Au final pour ma configuration il faut 3 ipfailover :

- 1 pour monter le tunnel VPN 1 sur mon serveur dédié ⇒ IP_FAILOVER_DEDIE_1
- 1 pour monter le tunnel VPN 2 sur mon serveur dédié ⇒ IP_FAILOVER_DEDIE_2
- 1 pour mon serveur à la maison ⇒ IP_FAILOVER_MAISON

Je prends uniquement des IP failover pour les VPN pour me permettre de les brancher facilement sur d'autres serveurs si besoin. C'est pour cela que je n'utilise jamais l'ip physique de mon serveur dédié.

Par ailleurs, il est important de noter qu'on a pas besoin de 2 ip pour monter deux tunnels VPN : il suffit d'utiliser des ports différents sur la même ip. Cela dit c'est un choix délibéré pour une raison simple : sur le seveur maison qui servira de routeur, on va devoir toucher ses tables de routage pour qu'il monte un tunnel VPN via la connexion ADSL et un autre via la 4G. Le plus simple est de jouer avec les tables de ***routage du serveur avec les ip de destination vers le dédié, une différente par connexion***. C'est plus simple.

Dernier détail important, comme online.net (et probablement n'importe quel hébergeur de dédié d'ailleurs) surveille son réseau pour éviter que n'importe quel équipement cause dessus, ***il faut également prendre une mac online.net (j'ai choisi la mac pour KVM bien que je pense que cela ne fasse pas de différence) avec l'ip failover associé au serveur maison***. Autrement, vous vous ferez couper le réseau puis que vous aurez votre serveur maison avec une adresse mac maison (non connue de online.net) qui va trainer sur leur réseau...

1.3. Configurer OpenVPN

1.3.1. Introduction

On est toujours sur le serveur à ce stade, il s'agit maintenant de préparer la **configuration serveur pour 2 tunnels VPN**.

Note importante : il est essentiel de comprendre que l'ordre de certaines opération, notamment au démarrage de la machine pourra changer la façon de préparer la configuration. Par exemple si OpenVPN démarre avant ou après le réseau ou inversement, on ne passera pas les commandes de la même manière car il y a un ordre logique. Par exemple, on utilise pas une interface VPN si elle n'est pas un minimum montée... Bref, j'indique ici **une technique qui peut ne fonctionner que dans le contexte ubuntu 14.04**. Il y a une manière de faire qui n'est pas due au hasard bien qu'il existe sans doute d'autres (meilleures peut-être) manières. J'ai juste choisi en fonction de ce que j'ai trouvé sur le net et ce que je trouvais le plus simple / pratique à gérer.

1.3.2. Préparation de la configuration

On crée le dossier de logs pour nos tunnels (j'ai fait comme j'ai trouvé à droite à gauche mais il y a sans doute mieux à faire) :

```
sudo mkdir /etc/openvpn/logs
```

On créé le dossier qui contiendra nos scripts maison :

```
sudo mkdir /etc/openvpn/scripts
```

On crée un premier script qui nous servira à mettre a jour l'état du lien sur l'interface VPN. Pourquoi ? Sachez que chaque interface contient un état qui permet de savoir si le lien réseau est OK ou KO (en

gros branché ou débranché). **C'est une chose qui va être très importante pour notre bonding car c'est la dessus que le bonding se base pour savoir si une interface réseau doit être utilisée pour acheminer du traffic ou alors mise de côté car elle ne fonctionne pas**. C'est utile pour la tolérance aux pannes qui est un des intérêts du bonding. En agrégeant deux liens, si l'un tombe il vous reste l'autre.

```
sudo nano /etc/openvpn/up-link.sh
```

Le contenu du fichier :

```
#!/bin/sh

DEV=$1

/sbin/ip link set "$DEV" up
echo "$(date) <----> $DEV LINK UP CALLED">> /etc/openvpn/logs/events
```

On crée ensuite le script complémentaire :

```
sudo nano /etc/openvpn/scripts/down-link.sh
```

Et le contenu du fichier :

```
#!/bin/sh

DEV=$1

/sbin/ip link set "$DEV" down
echo "$(date) <----> $DEV LINK DOWN CALLED">> /etc/openvpn/logs/events
```

1.3.3. Le fichier de configuration OpenVPN (enfin !)

On cree le fichier de configuration du premier serveur pour le premier tunnel :

```
sudo nano /etc/openvpn/server1.conf
```

Le fichier de configuration du premier serveur ressemble à cela chez moi *(les parties à adapter à votre configuration commencent par TO_CHANGE)* :

```
# Source :
http://web.archive.org/web/20130514042137/http://blog.geekass.net/2012/01/22
/agregation-de-liens-vpn
# Source :
http://www.debian-fr.org/install-openvpn-en-mode-bridge-t4376.html
# Source :
http://blog.héry.com/article6/cluster-proxmox-distant-construction-du-cluste
r-openvpn
# Source : https://help.ubuntu.com/community/OpenVPN
```

```
#
# Configuration for main tap
# To be used with fast connexion (4G)
#

#
# Main configuration
#
local TO_CHANGE_IP_FAILOVER_DEDIE_1
port TO_CHANGE_PORT_1
proto tcp-server
dev tap1

#
# Security configuration
#
secret static.key

#
# Tunnel main options
#
comp-lzo
persist-key
persist-tun

#
# Tunnel options to have correct link status update with scripts
#
keepalive 1 5
up-restart

#
# Scripts to enable correct link status update
#
down "/etc/openvpn/scripts/down-link.sh tap1"
ipchange "/etc/openvpn/scripts/up-link.sh tap1"

#
# Logs configuration
#
log logs/server1.log
verb 4
```

Eléments importants :

- **keepalive 1 5** permet de régler le maintient d'activité sur le VPN. S'il n'y a pas de traffic, un ping sera émis toutes les 1 sec et s'il n'y a pas du tout de traffic au bout de 5 sec, le tunnel est considéré comme perdu et cherchera à redémarrer.
- **up-restart** permet de faire en sorte que les “hooks” comme down (et les scripts associés down-link.sh) soient appelés en cas de restart de tunnel (c'est à dire en cas de défaillance) et

non pas juste une fois à l'arrêt d'OpenVPN.
- **down** (combiné a up-restart) permet d'appeler un script quand le tunnel tombe
- **ipchange** permet d'appeler un script quand le tunnel est remonté

Pour le deuxième serveur on part du premier :

```
sudo cp -av /etc/openvpn/server1.conf /etc/openvpn/server2.conf
```

Et pour le contenu il suffit d'adapter :

- TO_CHANGE_IP_FAILOVER_DEDIE_1 ⇒ TO_CHANGE_IP_FAILOVER_DEDIE_2
- TO_CHANGE_PORT_1 ⇒ TO_CHANGE_PORT_2
- tap1 ⇒ tap2

1.3.4. up-restart down et ipchange sont dans un bateau

Un petit aparté sur ces éléments et les scripts qui vont avec. C'est un truc très très con que j'ai du fouiller et bricoler au fur et a mesure.

En fait, les interfaces montées avec OpenVPN (ici tap1 et tap2) ne sont pas mises à jour correctement du point de vue link. C'est à dire que **quand le tunnel VPN tombe, l'interface associée (mettons tap1 pour le tunnel 1) reste toujours up !!!** Ce qui est une connerie fondamentale puisque les outils tels que le bonding se servent de cet état pour déterminer si le traffic peut être envoyé sur l'interface...

Bref, par défaut vous allez monter votre bonding avec 2 liens en répartition de charge et si un tunnel tombe, votre bonding continuera à envoyer du traffic sur le lien mort ce qui vous fera une perte de 50% des paquets. Bref, l'intérêt du bonding devient vraiment minable...

Donc voilà, avec les petites ruses précédentes, je m'arrange pour détecter quand le tunnel tombe / remonte et mettre a jour manuellement l'état du lien afin qu'il puisse être correctement mis à jour up ou down.