# Serveur de cache APT / cron-apt

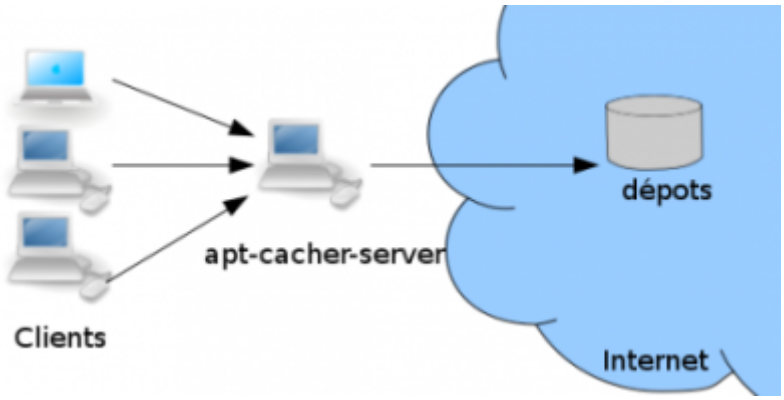
Script d'auto configuration d'un **serveur de cache pour APT, côté serveur et client**. Installation d'un **cron-apt récupérant les mises à jour la nuit** et **installant automatiquement les mises à jour de sécurité** des repo. présentes sur votre système. Gestion de la configuration aussi bien niveau **installation client / serveur**.

Ce script peut être installé via un compte **root** (base debian) mais également via **sudo** (base ubuntu, mint...). Il est écrit en **python3** et doit être lancé avec les droits administrateur.

Utilité dans le cadre de la gestion d'un parc de PC. Un PC est défini comme **serveur** et sera le seul à utiliser la bande passante de l'Internet pour récupérer les mises à jour. **Les clients** se connectent, eux, via ce PC uniquement pour **apt** et donc utilisent le réseau local. **Le parc de client** peut être **hétérogène** (au niveau de la distribution mais aussi de leurs versions : Debian Stable / SID / Mint / Lubuntu / Mandriva ...) du moment qu'ils utilisent **des paquets au format Debian (.deb)**.

La mise à jour du cache se fera depuis le serveur, mais également dès l'accès à ce serveur par un client.

*Apt-Cacher NG est un mandataire de cache pour le téléchargement de paquets depuis des dépôts de logiciels dans le style de Debian (ou d'autres types).. Le principe est qu'une machine centrale héberge le mandataire pour un réseau local. Les clients règlent leur configuration d'APT pour télécharger sur cette machine. Apt-Cacher NG conserve une copie de toutes les données utiles transitant à travers lui et, quand une requête similaire est faite, la copie en cache des données est délivrée sans être téléchargée à nouveau.*




## Les avantages d'APT-CACHER-NG

- apt-cacher-ng est un gain de temps
- apt-cacher-ng limite l'utilisation de la bande passante
- apt-cacher-ng permet d'intégrer des images ISO (DVD) et des importations de cache apt

## Configurations du script

### Configuration, côté serveur

Installation des paquets :

- apt-cacher-ng
- cron-apt

Voir config **cron-apt** plus bas.

### Configuration, côté client

- cron-apt

## Le script

Ci dessus la version 2.0.0

Mais préfèrable de suivre la version du GIT : https://github.com/CyrilleBiot/scripts/

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" Script d'installation et de configuration du serveur de cache apt
    apt-cacher-ng soit en tant que serveur (ajout du paquet sur le système
    soit en tant que client (cration d'un fichier de proxy apt)
    Possibilité pour les clients de choisir le port d'écoute du serveur
    Installation de cron-apt avec configuration spécifique
    pour installation automatique des mises à jour de sécurité des repo
    présents sur le système
"""

__author__ = "Cyrille BIOT"
__copyright__ = "Copyleft"
__credits__ = "Cyrille BIOT"
__license__ = "GPL"
__version__ = "2.0.0"
__date__ = "2020/02/05"
__maintainer__ = "Cyrille BIOT"
__email__ = "cyrille@cbiot.fr"
__status__ = "Devel"

import os, re, sys, platform
import nmap, subprocess, socket




def baseDebian():
```

```
    """
    Fonction permettant de connaitre le Systeme d'exploitant faisant tourner
le script
    Ou DEBIAN ou UBUNTU pour savoir si on utilise su ou sudo
    Retourne une variable de type string (admin)
    :return: admin soit 'debian' (root), soit 'ubuntu' (sudo)
    """
    # Ubuntu ou DEBIAN
    if 'Debian' in platform.version():
        # Si DEBIAN, verif si root lance le script
        print('Vous utilisez un système Debian (su pour administration).')
        if not os.geteuid() == 0:
            sys.exit("Seul le root peut lancer ce script. Nécessite
privilèges administrateur.")
        distrib = 'debian'
    else:
        if not os.geteuid() == 0:
            print("Ce programme requiert un lancement via 'sudo'")
            sys.exit("Ce programme doit être lancé avec les droits
administrateur.\nUtiliser sudo LeScript.py")
        print('Vous utilisez un système non Debian (sudo pour
administration).')
        distrib = 'ubuntu'
    return distrib

def installPackage(package, debianUbuntu):
    """
    Fonction installant un package debian ou ubuntu
    :param package: le nom du paquet à installer
    :param debianUbuntu: soit 'debian' / soit 'ubuntu'
    :return: None
    """
    retval = subprocess.call(['which', package])
    if retval != 0:
        print("Le package {} n'est pas intallé.
Installation...".format(package))

        # Paramètres de l'install
        cmdInstall = ['apt-get', 'install', package, '-y']
        cmdUpdate = ['apt-get', 'update']

        # Adaptation système Ubuntu
        if debianUbuntu == 'ubuntu':
            cmdInstall.insert(0, 'sudo')
            cmdUpdate.insert(0, 'sudo')

        # On installe le paquet
        subprocess.run(cmdInstall)
        #subprocess.run(cmdUpdate)
    else:
        print('Le package {} est déjà présent sur votre
```

```
système.'.format(package))

    return None

def installServeur(ip, port,distrib):
    """
    Fonction installant le serveur de cache apt-cacher-ng
    :param ip: IP du Serveur
    :param port: interger port ACN
    :param distrib: Ubuntu ou Debian
    :return: None
    """

    # Installation du serveur
    installPackage('apt-cacher-ng',distrib)

    # Affichage Informations
    print("================================================")
    print("Le serveur de cache est dès lors opérationnel")
    print("Le port d'écoute est : {}".format(port))
    print("Page d'aministration : http://{}:{}/acng-report.html".format(ip,
port))
    print("Notez bien l'ip de votre serveur, elle vous sera indispensable
pour la configuration des clients.")
    print("L'IP du serveur est : {} ".format(ip))
    print("Indispensable : cette IP doit être FIXE (réglage sur votre BOX ou
serveur DHCP).")
    print("Cette machine est un serveur, mettre de ne l'arrêter. Les mises à
jour s'effectuant la nuit.")

    return None


def installClient(ipServeur,portACN):
    """
    Fonction installant un fichier de configuration apt pour les postes
clients
    Créer un fichier dans /etc/apt/apt.conf.d/ ayant pour nom 00aptproxyANC
    :param ipServeur:  ip du serveur ACN
    :param portACN: port d'écoute du serveur ACN
    :return: None
    """

    # COnfig IP serveur dans un fichier de proxy APT
    msgApt = 'Acquire::http::Proxy "http://' + ipServeur + ':' +
str(portACN) + '";\n'
    print(msgApt)
    dirInstall = '/etc/apt/apt.conf.d/'
    fileName = '00aptproxyANC'
    fileLocInstall = dirInstall + fileName
    fichier = open(fileLocInstall, "w")
```

```
    fichier.write(msgApt)
    fichier.close()
    return None



def portSelection(portACN):
    while True:
        try:
            portDefault = input("Utiliser le port par défaut 3142
(recommandé) ?. [Oui / Non]  ")
            if portDefault.lower() == 'oui':
                print('Port Serveur {}'.format(portACN))
                break

            elif portDefault.lower() == 'non':
                try:
                    portSelect = int(input("Saisir le port du serveur Apt-
Cacher-Ng. Entre 0 et 65 535. : "))
                    if -1 < portSelect < 65536:
                        print("Port sélectionné{}".format(portSelect))
                        portACN = portSelect
                        break
                except ValueError:
                    print("Oops!  Réponse incorrecte, ce n'est pas un nombre
compris dans la plage demandée.")
        except ValueError:
            print("Oops!  Réponse incorrecte... Réessayer...")

    print("Installation client sur port {}.".format(portACN))

def ipRecuperation():
    """
    Fonction récupérant l'adresse IPv 4 de la machine
    :return: l'ip de la machine lançant ce script
    """
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
    s.connect(('<broadcast>', 0))
    return s.getsockname()[0]



def ipTest(ip):
    """
    Fonction testant la validité d'une adresse IPv4
    :param ip: ip à tester
    :return: True si IP valide, False sinon
    """
    reg =
r"^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|
1[0-9]{2}|2[0-4][0-9]|25[0-5])$"
```

```
    if re.match(reg, ip):
        return True
    else:
        return False


def clientServeur():
    """
    Fonction déterminant s'il s'agit d'une installation de type Serveur ou
Client
    :return: Retourne une variable string soit client soit serveur
    """
    while True:
        try:
            choixInstall = input("Type d'installation (client/serveur) : ")
            if choixInstall.lower() in ['client', 'serveur']:
                print('Installation de type {}'.format(choixInstall))
                break
            else:
                print('Préciser : client OU serveur.')
                print('ATTENTION A LA CASSE. Pas de majuscule.')
        except ValueError:
            print("Oops!  Réponse incorrecte... Réessayer...")
    return choixInstall

def portStatus(ip, port):
    """
    Fonction de scanne d'un port d'une machine en fonction de son IP
    :param ip: IP de la machine à scanner
    :param port: port à scanner
    :return: Retourne True si port ouvert ou False si port fermé
    """

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(1)  #
    result = sock.connect_ex((ip, port))
    if result == 0:
        message = str(ip) + ' : Le port ' + str(port) + ' est ouvert.
Possibilié de serveur ACN.'
        print(message)
        return True
    else:
        message = str(ip) + ' : Le port est fermé. Code d\'erreur de retour;
' + str(result)
        message += '. Pas de serveur ACN'
        print(message)
        return False

def chercherServeurACN(ip,port):
    """
    Fonction recherchant
```

```
    :param ip: IP du client lançant le scan, permet de trouver un motif
réseau
    :param port: port à scanner (port ACN)
    :return: retourne une liste contenant les IP possibles des machines
ayant port spécifié ouvert
    """
    ipModele = ''
    listeHosts = []
    ipServeurACN = []


    # Création d'un motif pour le scan reseau
    ipSplit = ip.split('.')
    for i in range (0,3):
        ipModele += ipSplit[i] + '.'
    ipModele += '0'

    # debug
    print('=' * 40)
    print('Votre machine possède l\'ip {}.\r\nLe motif de scan sera donc :
{}'. format(ip,ipModele))

    # Scan reseau à la recherche de clients
    nm = nmap.PortScanner() # instantiate nmap.PortScanner object
    nm.scan(hosts=ipModele+'/24', arguments='-n -sP')
    for host in nm.all_hosts():
        print('----------------------------------------------------')
        print('Host : %s (%s)' % (host, nm[host].hostname()))
        print('State : %s' % nm[host].state())
        # Creation d'un mappage reseau
        listeHosts.append(host)

    # Sca, port ACN des clients
    print('=' * 40)
    print('Résultats du scan réseau : (True si port Apt-cache-server
trouvé.')
    # Pour chacune des machines du réseau, on teste le port d'ACN (par
defaut 3142
    for i in listeHosts:
        testPort = portStatus(i, port)
        # Si réponse True, c'est le serveur
        if testPort == True:
            ipServeurACN.append(i)
            message = 'Eventuel Serveur ACN.'
        else:
            message = 'Pas de port ACN ouvert'
        print(i, ' : ', testPort, '. ', message)

    return ipServeurACN

def validerIpServeurACN(listIp):
```

```
    """
    Fonction recupérant la liste des machines susceptibles d'être serveur
ACN
    Teste de cette liste pour valider ces IP ou les infirmer
    :param listIp: liste contenant les IP des machines écoutant le port ACN
    :return: IP de la machine sélectionnée comme serveur ACN
    """
    if len(listIp) == 0:
        sys.exit('Aucun serveur ACN de trouver. Merci de vérifier son
installation.\r\n'
                 'Relancer ce script sur la machine serveur.\r\n'
                 'Et sélectionner "Installation Serveur"\r\n')
    elif len(listIp) == 1:
        print('Serveur ACN possible : ',listIp[0])
        while True:
            try:
                ouiNon = input("Valider ce choix ? (Oui / Non) ")
                if ouiNon.lower() == 'oui':
                    print('IP du serveur : ', listIp[0] )
                    ipServeur = listIp[0]
                    break
                elif ouiNon.lower() == 'non':
                    sys.exit('Revoir la configuration du serveur.\n'
                             'Et relancer ce script.\n'
                             'Aucune machine disponible dans le reseau
actuellement '
                             'avec ce port d\'ouvert')
            except ValueError:
                print("Oops!  Réponse incorrecte... Réessayer... [Oui / Non
]")
        # Valider l'ip unique
    else:
        print('Plusieurs machines pouvant être des serveurs ACN')
        print('Veuillez sélectionner une ip, merci :')
        for i in enumerate(listIp):
            print('Choix ', i[0] + 1, ' : ', i[1])
        # Installation client
        while True:
            try:
                ipServeur = input("Saisir l'IP du Serveur :")
                if ipTest(ipServeur) is True and ipServeur in listIp:
                    break
            except ValueError:
                print("Oops!  Réponse incorrecte... Réessayer...")
    return ipServeur


def installCronApt(distrib):
    """
    Fonction Recuperation des entrées des mises à jour de sécurité dans
     dans les divers sources.list possibles
```

```
     Et création d'un sources.list basé que sur ces entrées (security)
     Le fichier est propre à primtux. Donc si existe, on le régénère sinon
on le crée
     Et envoi mail sur root
    :param distrib: Ubuntu ou Debian
    :return: None
    """

    mailRoot = 'root'
    aptSecurity = "find /etc/apt -type f -name '*.list' " \
                  "| xargs cat " \
                  "|  grep -v \"^#\" | grep security"

    # Installation de cron-apt
    installPackage('cron-apt',distrib)

    # Création sources.list spécial sécurité
    log = open('/etc/apt/sources.list.d/security-primtuxACN.list', 'w')
    log.write('# Security Update. For Primtux Apt-cacher-ng.\n')
    log.flush()
    c = subprocess.call(aptSecurity, stdout=log, stderr=log, shell=True)

    # Configuration d'une action dans la conf de cron-apt
    # /etc/cron-apt/action.d/5-primtuxACN-security
    fichier = open('/etc/cron-apt/action.d/5-primtuxACN-security', "w")
    fichier.write("upgrade -y -o APT::Get::Show-Upgraded=true\n")
    fichier.write("OPTIONS=\"-o quiet=1 -o APT::Get::List-Cleanup=false -o "
                  "Dir::Etc::SourceList=/etc/apt/sources.list.d/security-
primtuxACN.list "
                  "-o Dir::Etc::SourceParts=\\\"/dev/null\\\"\"\n")
    fichier.write("MAILTO=\"{}\"\n".format(mailRoot))
    fichier.write("MAILON=\"always\"\n")
    fichier.close()

    print("Dès lors, le système installera les mises à jour de sécurité,
toutes les nuits  à 4 heures.")

    return None



def main():
    """
    Lancement du script
    :return: None
    """
    # Défnition du port par defaut d'ACN
    portACN = 3142
    # Recupere le type de distribution faisant tourner le script
    distrib  = baseDebian()
```

```
    choixInstall = clientServeur()
    if choixInstall.lower() == 'serveur':
        ipServeur = ipRecuperation()
        installServeur(ipServeur, portACN, distrib)
    else:
        # Installation client
        portSelection(portACN)
        ip = ipRecuperation()
        ipServeur = chercherServeurACN(ip, portACN)
        ipServeur = validerIpServeurACN(ipServeur)
        installClient(ipServeur,portACN)

    # Que ce sont pour l'un ou l'autre, install cron-apt auto securité
    installCronApt(distrib)
    return None

"""
 Boucle main()
"""
if __name__ == "__main__":
    # execute only if run as a script
    main()
```

## Liens

- Le paquet sous DEBIAN SID
- Homepage Apt-cacher-ng

From:
https://cbiot.fr/dokuwiki/ - **Cyrille BIOT**

Permanent link:
**https://cbiot.fr/dokuwiki/python:acn-py-installer?rev=1581205475**

Last update: **2020/02/08 23:44**