Gandi, autohébergement et IP dynamique : la solution, c'est Gandyn !

V2.1.1 du 10 juillet 2018.

Table des matières

Introduction :

Si comme moi vous vous auto-héberger, il y a de fortes chances pour que votre fournisseur d'accès (Orange, Free…) ne vous fournisse qu'une IPv4 dynamique ! Bref, une IP qui change régulièrement, souvent au reboot de votre box… Et qui bloque ainsi l'accès depuis l'extérieur à vos services… (je n'aborderai pas ici l'IPv6 car encore trop peu mise en place par les fournisseurs d'acès Internet français).

Par conséquent, quand on débute, trouver une parade relève parfois du défi et l'on finit souvent par les solutions suivantes :

- ✔ DNSexit
- ✔ No-IP
- ✔ ChangeIP
- ✔ DynDNS (en italien)

Mais aucune n'est réellement gratuite et tout se complique rapidement. Pourtant, le problème n'est pas si complexe au fond : « il s'agit de détecter l'adresse IP publique de son serveur puis de la transmettre à son registrar afin qu'il modifie la zone DNS afin de refaire le lien entre le nom de domaine que vous avez acheté et votre IP qui vient d'être renouvelée. » Avec ce nouveau tuto, je vais vous expliquer comment faire cette manipulation « à la main » puis à l'automatiser à intervalles réguliers.

C'est cette seconde option que je vais vous détailler ici parce qu'au delà de libérer l'utilisateur, la démarche est particulièrement formatrice.

Bonne lecture !

Principe général et contexte :

Voici la situation de départ :

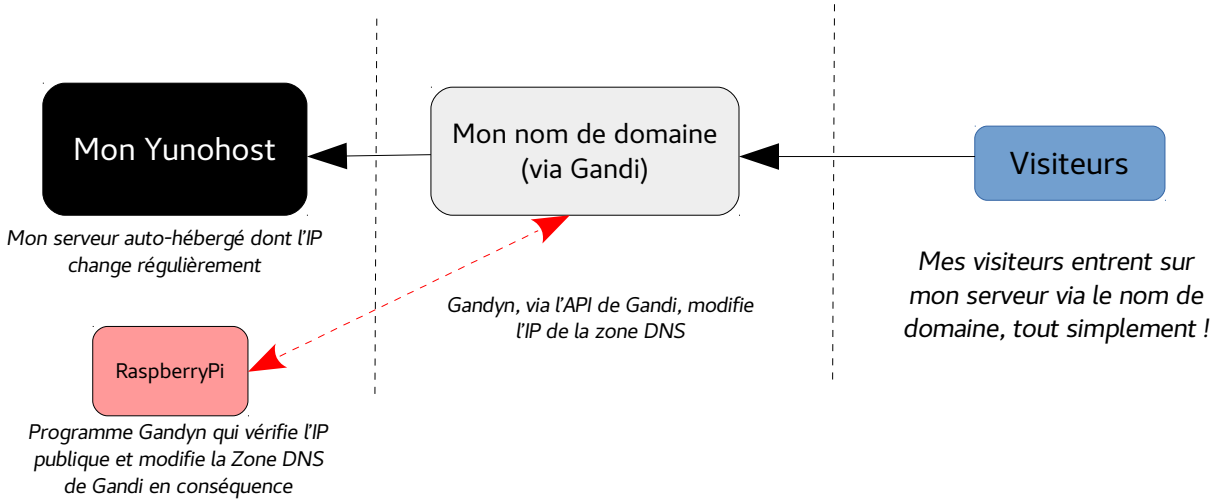
- ✔ Serveur auto-hébergé (physiquement chez moi!) configuré à partir de la solution Yunohost.
- ✔ Noms de domaines achetés chez Gandi.
- ✔ Fournisseur d'accès Orange Fibre, avec IP dynamique (IP changeant toutes les 3 à 4 semaines en moyenne).

Rappel pour les débutants : afin de faire le lien entre votre nom de domaine et l'adresse IP publique de votre serveur, il faut configurer la « Zone DNS » via l'interface de Gandi. En clair, il faut, dans ma configuration, copier-coller le fichier de configuration DNS de Yunohost (disponible via l'interface web d'administration de votre serveur) dans l'interface web de la zone DNS de Gandi.

Par conséquent, le problème est le suivant : à chaque changement de l'adresse IP publique de votre serveur, il faut refaire la configuration !

Heureusement, avec un peu d'huile de coude, et surtout grâce à l'API Gandi, il suffit de configurer un petit script Python (Gandyn, pour ne pas le citer) qui fera cette reconfiguration automatiquement dès que l'adresse IP changera !

Schéma d'ensemble :



Matériel utilisé :

- Un Raspberry Pi B *(machine sur laquelle les automatisations seront configurées)*

Sources m'ayant aidé à la configuration :

- Le site https://yunohost.org/#/dns_dynamicip_fr
- Le site https://github.com/Chralu/gandyn
- Le site https://www.raspberrypi.org/

Et un peu de motivation…

Étape n°1 : installer Raspbian sur le Raspberry Pi.

La procédure étant déjà détaillée un peu partout, je me permets de vous renvoyer sur le site du Raspberry Pi :
https://www.raspberrypi.org/documentation/installation/installing-images/README.md

Étape n°2 : mettre le système à jour.

➔ ~$ sudo apt-get update

➔ ~$ sudo apt-get upgrade

➔ ~$ sudo apt-get dist-upgrade

➔ ~$ sudo rpi-update

Étape n°3 : installer Gandyn

➔ ~$ wget -O gandyn.tar.gz https://github.com/Chralu/gandyn/tarball/<version>
➔ ~$ tar xvzf gandyn.tar.gz
➔ ~$ cd gandyn/src/
➔ ~$ sudo python setup.py install
➔ ~$ sudo nano /home/pi/gandyn/src/gandyn.py

Voici le script python à compléter avec votre clé API Gandi et votre nom de domaine :

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-
import os
import sys
import getopt
import xmlrpc.client
import logging
import ipretriever
import ipretriever.adapter
API_KEY = 'votre_clé_API-Gandi'
DOMAIN_NAME = 'domaine.fr'
TTL = 300

RECORD = {'type':'A', 'name':'@'}

LOG_LEVEL = logging.INFO
LOG_FILE = 'gandyn.log'
```

```
class GandiDomainUpdater( object ):
  """Updates a gandi DNS record value."""
  def __init__(self, api_key, domain_name, record):
    """Constructor

    Keyword arguments:
    api_key -- The gandi XML-RPC api key. You have to activate it on gandi website.
    domain_name -- The domain whose record will be updated
    record -- Filters that match the record to update
    """
    self.api_key = api_key
    self.domain_name = domain_name
    self.record = record
    self.__api = xmlrpc.client.ServerProxy('https://rpc.gandi.net/xmlrpc/')
    self.__zone_id = None

  def __get_active_zone_id( self ):
    """Retrieve the domain active zone id."""
    if self.__zone_id == None :
      self.__zone_id = self.__api.domain.info(
        self.api_key,
        self.domain_name
        )['zone_id']
    return self.__zone_id

  def get_record_value( self ):
    """Retrieve current value for the record to update."""
    zone_id = self.__get_active_zone_id()
    return self.__api.domain.zone.record.list(
      self.api_key,
      zone_id,
      0,
      self.record
      )[0]['value']

  def update_record_value( self, new_value, ttl=300 ):
    """Updates record value.

    Update is done on a new zone version. If an error occurs,
    that new zone is deleted. Else, it is activated.
    This is an attempt of rollback mechanism.
    """
    new_zone_version = None
    zone_id = self.__get_active_zone_id()
    try:
      #create new zone version
      new_zone_version = self.__api.domain.zone.version.new(
        self.api_key,
        zone_id
        )
      logging.debug('DNS working on a new zone (version %s)', new_zone_version)
      record_list = self.__api.domain.zone.record.list(
        self.api_key,
        zone_id,
        new_zone_version,
        self.record
        )
      #Update each record that matches the filter
      for a_record in record_list:
        #get record id
        a_record_id = a_record['id']
        a_record_name = a_record['name']
        a_record_type = a_record['type']

        #update record value
        new_record = self.record.copy()
        new_record.update({'name': a_record_name, 'type': a_record_type, 'value': new_value, 'ttl': ttl})
        updated_record = self.__api.domain.zone.record.update(
          self.api_key,
```

```
            zone_id,
            new_zone_version,
            {'id': a_record_id},
            new_record
            )
        except xmlrpc.client.Fault as e:
          #delete updated zone
          if new_zone_version != None :
            self.__api.domain.zone.version.delete(
              self.api_key,
              zone_id,
              new_zone_version
              )
          raise
        else:
          #activate updated zone
          self.__api.domain.zone.version.set(
            self.api_key,
            zone_id,
            new_zone_version
            )

    def usage(argv):
      print(argv[0],' [[-c | --config] <config file>] [-h | --help]')
      print('\t-c --config <config file> : Path to the config file')
      print('\t-h --help                 : Displays this text')

    def main(argv, global_vars, local_vars):
      try:
        options, remainder = getopt.getopt(argv[1:], 'c:h', ['config=', 'help'])
        for opt, arg in options:
          if opt in ('-c', '--config'):
            config_file = arg
            #load config file
            exec(
                compile(open(config_file).read(), config_file, 'exec'),
                global_vars,
                local_vars
            )
          elif opt in ('-h', '--help'):
            usage(argv)
            exit(1)
      except getopt.GetoptError as e:
        print(e)
        usage(argv)
        exit(1)

      try:
        logging.basicConfig(format='%(asctime)s %(levelname)-8s %(message)s', datefmt='%a, %d %b %Y %H:%M:%S', level=LOG_LEVEL,
    filename=LOG_FILE)
        public_ip_retriever = ipretriever.adapter.IPEcho()
        gandi_updater = GandiDomainUpdater(API_KEY, DOMAIN_NAME, RECORD)

        #get DNS record ip address
        previous_ip_address = gandi_updater.get_record_value()
        logging.debug('DNS record IP address : %s', previous_ip_address)

        #get current ip address
        current_ip_address = public_ip_retriever.get_public_ip()
        logging.debug('Current public IP address : %s', current_ip_address)

        if current_ip_address != previous_ip_address:
          #update record value
          gandi_updater.update_record_value( current_ip_address, TTL )
          logging.info('DNS updated')
        else:
          logging.debug('Public IP address unchanged. Nothing to do.')
      except xmlrpc.client.Fault as e:
        logging.error('An error occured using Gandi API : %s ', e)
```

```
    except ipretriever.Fault as e:
      logging.error('An error occured retrieving public IP address : %s', e)

main(sys.argv, globals(), locals())
```

- ➔ ~$ sudo chmod +x /home/pi/gandyn/src/gandyn.py
- ➔ ~$ sudo crontab -e

Exemple :
0 5 * * * /home/pi/gandyn/src/gandyn.py (pour une exécution de la tache à 5h du matin tous les jours.

Attention : il faudra répéter l'opération pour chacun des « RECORD » nécessaires. Pour Gandi, j'en ai donc 3 à configurer, c'est à dire 3 fichiers gandyn à préparer !

```
RECORD = {'type':'A', 'name':'@'}
RECORD = {'type':'A', 'name':'*'}
RECORD = {'type':'TXT', 'name':'@'}
```

Remarque très importante : il n'est pas possible de faire 2 « record » en même temps, donc prévoyez 5 min entre chacun ! (par exemple, un à 5h, le second à 5h05, le troisième à 5h10…).

Conclusion

Grâce à ce tuto, vous êtes désormais autonome et le seul coût financier à supporter sera celui du fonctionnement 24h/24 de votre petit RaspberryPi ne consommant qu'environ 3 watts/h (soit moins de 5€ en moyenne à l'année).