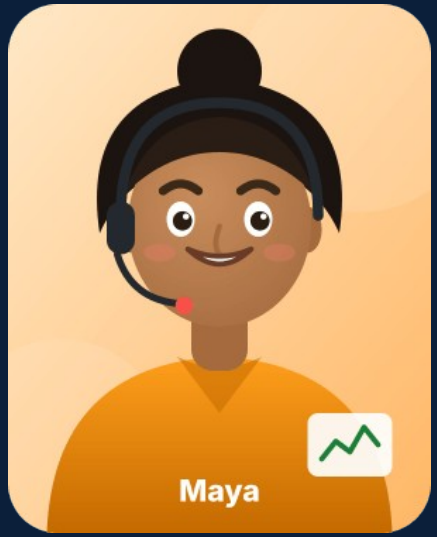
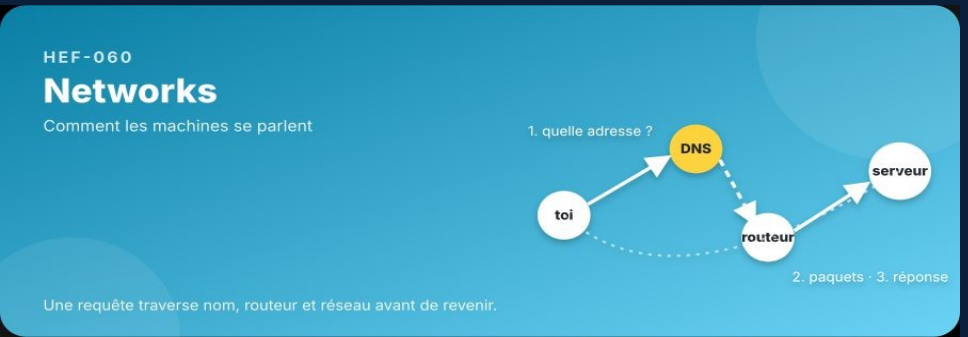
HEF-060 · MISSION 06 / 10

DNS : l'annuaire d'internet

Personne ne retient 93.184.216.34. On retient exemple.fr. Quelqu'un doit faire la traduction.



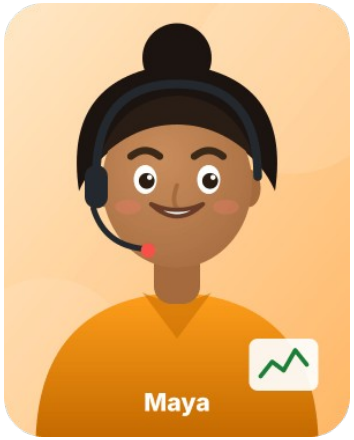


Badge : Bibliothécaire des noms

75 min · intermédiaire

L'histoire

HashBox déménage son site chez un autre hébergeur : nouvelle adresse IP. Kodi modifie un enregistrement, et pendant plusieurs heures, certains utilisateurs arrivent sur l'ancien serveur, d'autres sur le nouveau.



> *« C'est le DNS, dit Sam. Des copies de l'annuaire sont gardées partout, avec une durée de vie. Il faut apprendre à les gérer. »*

Tu comprends comment un nom devient une adresse, et pourquoi tant d'incidents commencent par « c'est le DNS ».

Ta mission

À la fin, je serai capable de…

1. expliquer le rôle du DNS et de la résolution de noms
2. décrire la hiérarchie racine, TLD, serveurs faisant autorité
3. citer les principaux types d'enregistrements (A, AAAA, CNAME, MX, TXT, NS)
4. expliquer le rôle du cache et du TTL
5. simuler une résolution complète en Python

Défi d'échauffement

Défi d'échauffement

Pour joindre une amie dont tu n'as que le nom, tu appelles les renseignements. Ils te demandent la ville, puis te donnent le numéro d'un service local, qui te renseigne sur la rue, puis sur le foyer. Quelles sont les **étapes** de cette recherche ? Si tu appelles souvent cette amie, que fais-tu pour ne pas refaire toute la recherche ? Quel risque s'il change de numéro ?

Le concept

Le problème

Les machines utilisent des **adresses IP**. Les humains préfèrent des **noms**. Le **DNS** (*Domain Name System*) est un système distribué qui traduit un **nom de domaine** en informations (adresse IP, serveur de mail…).

La hiérarchie

Un nom se lit **de droite à gauche** :

```
www . exemple . fr .
 |        |      |   └── la racine (le point final, souvent invisible)
 |        |      └────── domaine de premier niveau (TLD) : .fr, .com, .org
 |        └───────────── domaine : exemple
 └────────────────────── sous-domaine : www
```

Chaque niveau est géré par des serveurs différents :

Le concept

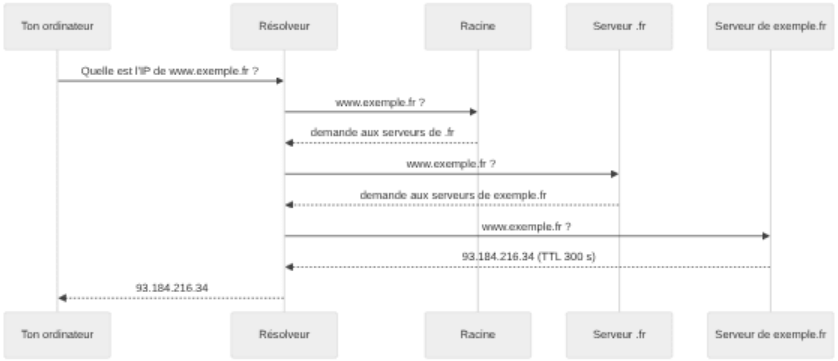
La hiérarchie (suite)

Niveau	Qui	Rôle
Racine (13 groupes de serveurs racines)	organismes mondiaux	sait qui gère chaque TLD
TLD (`.fr`, `.com`)	registre	sait quels serveurs gèrent chaque domaine
Serveur faisant autorité	le propriétaire du domaine (ou son hébergeur)	donne la réponse définitive

La résolution

Ton ordinateur ne parle pas directement à la racine. Il interroge un **résolveur récursif** (celui de ton fournisseur, ou `8.8.8.8`, `1.1.1.1`…) qui fait tout le travail :

La résolution (suite)



Description du schéma : l'ordinateur demande au résolveur, qui interroge successivement la racine, le serveur du TLD, puis le serveur faisant autorité, et renvoie la réponse finale à l'ordinateur.

Les types d'enregistrements

Type	Contenu	Exemple
A	adresse IPv4	`www.exemple.fr → 93.184.216.34`
AAAA	adresse IPv6	`www.exemple.fr → 2606:2800:…`
CNAME	alias vers un autre nom	`blog.exemple.fr → hebergeur.example.net`
MX	serveur de mail du domaine	`exemple.fr → mail.exemple.fr (priorité 10)`
NS	serveurs faisant autorité	`exemple.fr → ns1.hebergeur.net`
TXT	texte libre (vérifications, SPF)	`"v=spf1 include:…"`

Le concept

Le cache et le TTL

Chaque réponse porte un **TTL** (*Time To Live*, en secondes). Pendant ce temps, les résolveurs et les systèmes **gardent la réponse en cache**. Avantages : rapidité, moins de charge. Inconvénient : un changement met du temps à se propager.

Bonne pratique avant un déménagement : **baisser le TTL** (par exemple à 60 s) quelques jours avant, faire le changement, puis le remonter.

Résoudre un nom en Python

```
import socket

# getaddrinfo interroge le système (hosts, DNS...). 'localhost' est résolu localem
   ent.
```

Le concept

Résoudre un nom en Python (suite)

```
for famille, _, _, _, adresse in socket.getaddrinfo("localhost", 80, proto=socket.
   IPPROTO_TCP):
    print(famille.name, adresse[0])
```

🧠 ***Astuce*** *: quand un site « ne marche plus » pour certains seulement, pense cache DNS et TTL.*

Exemple pas à pas

Problème : simuler un résolveur avec cache et TTL.

```
ZONES = {
    "exemple.fr": {"A": {"www": ("93.184.216.34", 300), "api": ("93.184.216.40", 6
   0)},
                   "CNAME": {"blog": ("www.exemple.fr", 300)}},
}

class Resolveur:
    def __init__(self):
        self.cache = {}          # (nom, type) -> (valeur, expire_a)
        self.requetes_autorite = 0
        self.horloge = 0

    def avancer(self, secondes):
        self.horloge += secondes

    def _chercher_autorite(self, nom, type_):
```

Exemple pas à pas

Exemple pas à pas (suite)

```
        self.requetes_autorite += 1
        sous, _, domaine = nom.partition(".")
        zone = ZONES.get(domaine)
        if zone and sous in zone.get(type_, {}):
            return zone[type_][sous]
        return None

    def resoudre(self, nom, type_="A"):
        cle = (nom, type_)
        if cle in self.cache and self.cache[cle][1] > self.horloge:
            return self.cache[cle][0], "cache"
        reponse = self._chercher_autorite(nom, type_)
        if reponse is None and type_ == "A":
            alias = self._chercher_autorite(nom, "CNAME")
            if alias:
                valeur, _ = self.resoudre(alias[0], "A")
                return valeur, "via CNAME"
        if reponse is None:
            return None, "NXDOMAIN"
```

Exemple pas à pas (suite)

```
        valeur, ttl = reponse
        self.cache[cle] = (valeur, self.horloge + ttl)
        return valeur, "autorité"

r = Resolveur()
print(r.resoudre("www.exemple.fr"))        # ('93.184.216.34', 'autorité')
print(r.resoudre("www.exemple.fr"))        # ('93.184.216.34', 'cache')
r.avancer(301)
print(r.resoudre("www.exemple.fr"))        # TTL expiré : ('93.184.216.34', 'autor
   ité')
print(r.resoudre("blog.exemple.fr"))       # via CNAME
print(r.resoudre("inconnu.exemple.fr"))    # (None, 'NXDOMAIN')
print("requêtes à l'autorité :", r.requetes_autorite)
```

Lecture :

- la 2e requête est servie par le **cache** (aucune requête à l'autorité) ;

Exemple pas à pas (suite)

- après 301 secondes (TTL de 300 dépassé), on **réinterroge** l'autorité ;
- un **CNAME** redirige vers un autre nom qu'il faut à son tour résoudre ;
- un nom inexistant renvoie **NXDOMAIN**.

Impact d'un déménagement : tant que le TTL n'a pas expiré, certains résolveurs répondent encore avec l'ancienne adresse. Avec un TTL de 86 400 s (un jour), la transition peut durer 24 h.

À toi, avec aide

Un site `shop.exemple.fr` a un TTL de 3 600 s. Tu changes son adresse à 12:00.

1. Au pire, à quelle heure tous les résolveurs auront-ils l'information ?
2. Quel TTL aurais-tu dû configurer 48 h avant pour limiter l'impact à 2 minutes ?
3. Simule avec le `Resolveur` ci-dessus : un client a résolu le nom à 11:50 (TTL 3 600) ; que voit-il à 12:30 ? À 13:00 ?

Indices progressifs

Indice 1

Le pire cas : un résolveur a mis la réponse en cache juste avant le changement.

Indice 2

L'ancien enregistrement peut vivre jusqu'à (moment de la dernière mise en cache + TTL).

Indice 3

12:00 + 1 h = 13:00.

Réponse

1. Au pire à **13:00** (un résolveur a mis en cache à 11:59 avec TTL de 3 600 s).
2. Un TTL de **120 s** (2 minutes) appliqué au moins 48 h avant (pour que l'ancien TTL long ait expiré partout).
3. Le client voit l'**ancienne** adresse à 12:30 (cache valable jusqu'à 12:50) ; à 13:00 il a expiré, il obtient la **nouvelle**.

```
# horloge en secondes depuis 11:50 ; le cache expire à 11:50 + 3600 = 12:50
mise_en_cache, ttl = 0, 3600
for nom, t in (("12:30", 40 * 60), ("13:00", 70 * 60)):
    print(nom, "ancienne" if t < mise_en_cache + ttl else "nouvelle adresse")
```

À toi, en solo

Débogage DNS. Voici le contenu d'une zone `exemple.fr` :

```
www     A       203.0.113.10
@       MX 10   mail.exemple.fr
mail    A       203.0.113.20
blog    CNAME   www
shop    CNAME   shop.exemple.fr
api     A       203.0.113.30
api     CNAME   www
```

1. Trouve **deux erreurs** dans cette zone (indice : un CNAME ne peut pas coexister avec un autre enregistrement de même nom ; un CNAME ne doit pas se pointer lui-même).
2. Écris un validateur `valider(zone)` qui les détecte automatiquement.
3. Propose les corrections.

À toi, en solo · réponse

Pistes de correction

```
def valider(zone):
    # zone : liste de (nom, type, valeur)
    erreurs = []
    par_nom = {}
    for nom, type_, valeur in zone:
        par_nom.setdefault(nom, []).append((type_, valeur))
        if type_ == "CNAME" and (valeur == nom or valeur == f"{nom}.exemple.fr"):
            erreurs.append(f"{nom}: CNAME qui pointe vers lui-même")
    for nom, enregs in par_nom.items():
        types = [t for t, _ in enregs]
        if "CNAME" in types and len(types) > 1:
            erreurs.append(f"{nom}: CNAME ne peut pas coexister avec {sorted(set(t
   ypes) - {'CNAME'})}")
    return erreurs

zone = [
    ("www", "A", "203.0.113.10"), ("@", "MX", "mail.exemple.fr"), ("mail", "A", "2
   03.0.113.20"),
    ("blog", "CNAME", "www"), ("shop", "CNAME", "shop.exemple.fr"),
```

Pistes de correction (suite)

```
    ("api", "A", "203.0.113.30"), ("api", "CNAME", "www"),
]
for e in valider(zone):
    print(e)
```

Erreurs : `shop` est un alias de lui-même (boucle) ; `api` a à la fois un `A` et un `CNAME`.
Corrections : faire pointer `shop` vers une vraie cible (par exemple `www`) ; garder soit le `A`, soit le `CNAME` pour `api`.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
TTL trop long avant un changement	Anciens utilisateurs mal redirigés pendant des heures	Baisser le TTL à l'avance
CNAME avec d'autres enregistrements	Comportement imprévisible	Un nom = soit CNAME seul, soit autres types
Oublier le cache local	On teste avec un résultat périmé	Vider le cache ou interroger directement l'autorité
Croire que DNS est instantané	Propagation = expiration des caches	Prévoir une transition

Mini-boss

Mini-boss

Résolveur itératif complet. Simule la hiérarchie : serveurs racine, TLD et autorité sous forme de dictionnaires qui renvoient soit une **réponse**, soit une **délégation** (« demande à tel serveur »).

1. Écris `resoudre_iteratif(nom)` qui part de la racine, suit les délégations et affiche chaque étape.
2. Ajoute un cache avec TTL et un compteur de requêtes réseau.
3. Montre qu'après un premier appel, un deuxième appel ne fait plus de requête et qu'un appel pour un autre nom du même domaine n'interroge plus que le serveur d'autorité.

Mini-boss · réponse

Pistes de correction

```
SERVEURS = {
    "racine": {"fr": ("delegation", "serveur-fr")},
    "serveur-fr": {"exemple.fr": ("delegation", "ns-exemple")},
    "ns-exemple": {"www.exemple.fr": ("reponse", "203.0.113.10", 300),
                   "api.exemple.fr": ("reponse", "203.0.113.30", 60)},
}

class Resolveur:
    def __init__(self):
        self.cache, self.requetes, self.horloge = {}, 0, 0
        self.serveur_domaine = {}          # domaine -> serveur d'autorité (déléga
   tions mémorisées)

    def interroger(self, serveur, nom):
        self.requetes += 1
        table = SERVEURS[serveur]
        for suffixe, rep in table.items():
            if nom == suffixe or nom.endswith("." + suffixe):
                return rep
```

Mini-boss · réponse

Pistes de correction (suite)

```
        return ("nxdomain",)

    def resoudre(self, nom):
        if nom in self.cache and self.cache[nom][1] > self.horloge:
            return self.cache[nom][0], "cache"
        domaine = ".".join(nom.split(".")[-2:])
        serveur = self.serveur_domaine.get(domaine, "racine")
        while True:
            rep = self.interroger(serveur, nom)
            print(f"  {serveur} → {rep[0]}")
            if rep[0] == "delegation":
                serveur = rep[1]
                if serveur == "ns-exemple":
                    self.serveur_domaine[domaine] = serveur
            elif rep[0] == "reponse":
                self.cache[nom] = (rep[1], self.horloge + rep[2])
                return rep[1], "résolu"
            else:
                return None, "NXDOMAIN"
```

Mini-boss · réponse

Pistes de correction (suite)

```
r = Resolveur()
print(r.resoudre("www.exemple.fr"), r.requetes)
print(r.resoudre("www.exemple.fr"), r.requetes)
print(r.resoudre("api.exemple.fr"), r.requetes)
```

Premier appel : 3 requêtes (racine, TLD, autorité). Deuxième : 0 (cache). Autre nom du même domaine : 1 seule (on connaît déjà l'autorité).

Quiz éclair

Quiz éclair

1. Le DNS sert à :

A. chiffrer les connexions
B. traduire des noms en adresses et autres informations
C. attribuer des adresses MAC

2. Un enregistrement MX indique :

A. l'adresse IPv6
B. un alias
C. le serveur de mail d'un domaine

3. Le TTL d'une réponse DNS détermine :

A. combien de temps elle peut être gardée en cache

Quiz éclair (suite)

B. la vitesse de réponse
C. la taille du paquet

4. 🔁 *Révision mission 05* : TCP garantit :

A. la vitesse maximale
B. l'ordre et l'absence de perte des octets
C. la confidentialité

5. 🔁 *Révision mission 05* : UDP est adapté à :

A. les transferts de fichiers
B. les paiements
C. la voix et la vidéo en direct

Quiz éclair · réponse

Réponses

1. **B**.
2. **C**.
3. **A**.
4. **B**.
5. **C**.

Je réfléchis

1. Quel équivalent du DNS utilises-tu (contacts du téléphone, raccourcis) ?
2. Pourquoi un cache est-il à la fois un avantage et une source de problèmes ?

À retenir

À retenir

1. DNS : noms ↔ adresses et autres informations.
2. Hiérarchie : racine, TLD, serveur faisant autorité.
3. Résolveur récursif + cache.
4. Types : A, AAAA, CNAME, MX, NS, TXT.
5. Baisser le TTL avant un changement.

MISSION ACCOMPLIE

+100 XP

Badge débloqué : Bibliothécaire des noms

Prochaine mission : HTTP et les API

