GeoIP dans PmaControl : résolution IPv4 et IPv6 sans fichier mmdb à chaque requête

Aurélien LEQUOY · 15 avril 2026

PMACONTROL GEOIP IPV6 MARIADB PERFORMANCE ARCHITECTURE

Le problème

Quand vous supervisez 100+ serveurs MySQL/MariaDB répartis dans plusieurs datacenters et pays, une question revient à chaque affichage de la page principale : **où est ce serveur ?**

La réponse classique : ouvrir le fichier GeoLite2 `.mmdb` de MaxMind, faire un lookup pour chaque IP, et afficher le drapeau du pays. Simple... sauf que :

- Ouvrir un fichier `.mmdb` de 70 Mo **à chaque requête HTTP** est coûteux
- Avec 100 serveurs à résoudre, c'est 100 lookups fichier par page
- Le fichier est un arbre binaire optimisé pour la lecture séquentielle, pas pour du burst parallèle
- En PHP-FPM, chaque worker recharge le fichier indépendamment

Sur PmaControl, la page `server/main` se rafraîchit **toutes les secondes** en AJAX. Ouvrir le `.mmdb` 100 fois par seconde est un non-sens.

La solution : tout mettre dans MariaDB

L'idée est simple : **importer l'intégralité des ranges GeoLite2 dans une table MariaDB**, puis faire des lookups SQL classiques. Un `SELECT` avec un index est bien plus rapide qu'un parcours d'arbre binaire sur disque.

Le schéma

```
CREATE TABLE data_geoip (
  id INT AUTO_INCREMENT PRIMARY KEY,
  network_start VARBINARY(16) NOT NULL,
  network_end   VARBINARY(16) NOT NULL,
  country_iso   CHAR(2) NOT NULL DEFAULT '',
  country_name  VARCHAR(100) NOT NULL DEFAULT '',
  INDEX idx_network_start (network_start)
) ENGINE=InnoDB;
```

Le choix de `VARBINARY(16)` est crucial :

- **4 octets** suffisent pour IPv4 (32 bits)
- **16 octets** sont nécessaires pour IPv6 (128 bits)
- `VARBINARY(16)` stocke les deux uniformément
- `INET6_ATON()` convertit n'importe quelle IP (v4 ou v6) en binaire comparable

La requête de lookup

```
SELECT country_iso FROM data_geoip
WHERE network_start <= INET6_ATON('89.30.104.134')
  AND network_end   >= INET6_ATON('89.30.104.134')
LIMIT 1;
```

Résultat : `FR` (France). Temps d'exécution : **< 1 ms**.

La même requête fonctionne pour IPv6 :

```
SELECT country_iso FROM data_geoip
WHERE network_start <= INET6_ATON('2001:4860:4860::8888')
  AND network_end   >= INET6_ATON('2001:4860:4860::8888')
LIMIT 1;
```

Résultat : `US` (United States — c'est le DNS public de Google).

L'import : itérer l'espace IP

IPv4 : de 0.0.0.0 à 255.255.255.255

L'espace IPv4 fait 2^32 = 4,3 milliards d'adresses. On ne les parcourt pas une par une — on utilise `getWithPrefixLen()` du reader MaxMind qui retourne le CIDR complet pour chaque adresse :

```
$ip = 0;
while ($ip <= 4294967295) {
    [$record, $prefixLen] = $reader->getWithPrefixLen(long2ip($ip));

    // Calculer la fin du réseau
    $networkSize = 1 << (32 - $prefixLen);
    $networkEnd = $ip + $networkSize - 1;

    if ($record && !empty($record['country']['iso_code'])) {
        // INSERT dans data_geoip
    }

    // Sauter tout le bloc CIDR
    $ip = $networkEnd + 1;
}
```

Le résultat : **~650 000 ranges** importées en quelques secondes. Chaque range couvre un bloc CIDR entier (ex: `89.30.104.0/22` → 1024 adresses en une seule ligne).

IPv6 : l'espace 2000::/3

L'espace IPv6 fait 2^128 adresses — impossible de le parcourir comme IPv4. Mais les adresses publiques routables vivent dans le bloc `2000::/3` (Global Unicast), et les allocations GeoIP sont typiquement en `/32` à `/48`.

Le principe est le même : avancer par la taille du préfixe retourné. La différence : on travaille avec des adresses binaires de 16 octets.

```
$current = inet_pton('2000::');
$end6    = inet_pton('3fff:ffff:ffff:ffff:ffff:ffff:ffff:ffff');
```

```
while ($current <= $end6) {
    [$record, $prefixLen] = $reader->getWithPrefixLen(inet_ntop($current));
    $endBin = binNetworkEnd($current, $prefixLen);

    if ($record && !empty($record['country']['iso_code'])) {
        // INSERT avec UNHEX(bin2hex(...))
    }

    $current = binIncrement($endBin); // +1 en arithmétique 128 bits
}
```

L'arithmétique binaire 128 bits est implémentée en PHP pur (pas de GMP requis) :

```
// Incrémenter une adresse IPv6 de 1
function binIncrement(string $bin): string|false
{
    $bytes = unpack('C16', $bin);
    for ($i = 15; $i >= 0; $i--) {
        $bytes[$i]++;
        if ($bytes[$i] <= 255) return pack('C16', ...$bytes);
        $bytes[$i] = 0; // carry
    }
    return false; // overflow
}
```

Le résultat en production

Volumes

Table	IPv4	IPv6	Total
`data_geoip` (country)	~650K	~180K	~830K ranges
`data_geoip_city` (city)	~3.7M	~1.2M	~4.9M ranges

Performance

Opération	Temps

Import country (IPv4 + IPv6)	~30 secondes
Import city (IPv4 + IPv6)	~5 minutes
Lookup 1 IP	< 1 ms
Lookup 100 IPs (page server/main)	~15 ms total
Refresh AJAX (1x/seconde)	négligeable

Affichage

Sur la page principale de PmaControl, chaque serveur affiche son drapeau emoji à côté de l'IP :

```
□□□ 89.30.104.134:3306    PIXID-MDB-MASTER1
□□□ 136.243.1.1:3306       Hetzner-Slave
□□□ 210.171.224.1:3306     NTT-Tokyo
□□□ 8.8.8.8:3306           Google-Test
```

Les IPs privées (10.x, 172.16-31.x, 192.168.x, 127.x) n'ont pas de record GeoLite2 — le drapeau est simplement absent.

Mise à jour

MaxMind met à jour GeoLite2 chaque semaine. Pour rafraîchir :

```
# Télécharger le nouveau .mmdb dans data/
# Puis relancer l'import :
php App/Webroot/index.php server loadGeoip       # country (~30s)
php App/Webroot/index.php server loadGeoipCity   # city (~5min)
```

L'import fait un `TRUNCATE` puis réinsère tout. Pas besoin de diff ou de migration — c'est un cache jetable.

Table city : aller plus loin

La table `data_geoip_city` ajoute région, ville, coordonnées GPS et fuseau horaire :

```
SELECT country_iso, region_name, city, latitude, longitude, time_zone
FROM data_geoip_city
```

```
WHERE network_start <= INET6_ATON('136.243.1.1')
  AND network_end   >= INET6_ATON('136.243.1.1')
LIMIT 1;
```

Résultat : `DE | Saxony | Falkenstein | 50.4779 | 12.3713 | Europe/Berlin`

Cela ouvre la porte à une cartographie des serveurs, à la détection de latence inter-datacenter, ou simplement à un affichage plus riche dans l'interface.

Pourquoi pas juste un LEFT JOIN ?

On pourrait vouloir faire un `LEFT JOIN data_geoip g ON g.network_start <= INET6_ATON(s.ip) AND g.network_end >= INET6_ATON(s.ip)` directement dans la requête serveur. Le problème : avec 650K ranges, ce range-join est coûteux pour l'optimiseur. On préfère N lookups individuels (1 par IP unique) qui sont instantanés grâce à l'index.

Conclusion

En important les données GeoLite2 dans MariaDB, on élimine la dépendance au fichier `.mmdb` à chaque rendu de page. Le lookup devient un `SELECT` indexé à < 1 ms, compatible IPv4 et IPv6, et la mise à jour est un simple `TRUNCATE + INSERT` hebdomadaire.

Le code source est disponible sur GitHub — contributions bienvenues.