TP 02 – Redis

Mise en cache d’une application MongoDB avec Redis

---

1. Objectifs du TP

L’objectif de ce second travail pratique est d’introduire un **système de cache** dans l’application développée lors du TP1, afin d’améliorer :

- les performances
- la scalabilité
- la charge sur la base MongoDB

Les objectifs précis étaient les suivants :

- Découvrir Redis et ses cas d’usage
- Mettre en place Redis via Docker
- Se connecter à Redis depuis PHP
- Mettre en cache les données pertinentes
- Gérer l’invalidation du cache
- Mesurer l’intérêt du cache (préparation aux tests de charge)

---

2. Présentation de Redis

Redis (Remote Dictionary Server) est une base de données **clé-valeur en mémoire**, extrêmement rapide.

Ses principales caractéristiques :

- stockage en mémoire (RAM)
- très faible latence
- persistance optionnelle
- structures de données avancées (strings, hash, list, set, etc.)

Dans ce TP, Redis est utilisé comme :

> **cache applicatif**, placé entre PHP et MongoDB.

---

3. Mise en place de Redis avec Docker

Redis est intégré à l’infrastructure existante via Docker Compose.

3.1 Service Redis

```
tpmongo-redis:
    container_name: "tpmongo-redis"
    image: redis:7.0.5
```

```
    ports:
        - 6379:6379
    networks:
        - local
```

3.2 Interface graphique Redis Commander

```
tpmongo-redis-commander:
    image: rediscommander/redis-commander:latest
    ports:
        - 8081:8081
    environment:
        - REDIS_HOSTS=local:tpmongo-redis:6379
```

Cette interface permet :

- de visualiser les clés
- de vérifier les TTL
- de debugger le cache

4. Configuration et connexion Redis en PHP

La connexion Redis est centralisée dans init.php.

```
function getRedisClient(): ? PredisClient
{
    if (!isset($_ENV['REDIS_ENABLE']) || strtolower($_ENV['REDIS_ENABLE'])
!== 'true') {
        return null;
    }

    $host = $_ENV['REDIS_HOST'];
    $port = (int)($_ENV['REDIS_PORT']);

    try {
        $redis = new PredisClient([
            'scheme'   => 'tcp',
            'host'     => $host,
            'port'     => $port,
        ]);
        if ($redis->ping() == 'PONG') {
            return $redis;
        } else {
            throw new Exception("Redis ping a échoué.");
        }
    } catch (Exception $e) {
        error_log("Erreur de connexion Redis : " . $e->getMessage());
        return null;
    }
```

```
}
```

Cette fonction :

- vérifie si Redis est activé
- établit la connexion
- teste la disponibilité avec un PING
- retourne null si Redis est indisponible

5. Choix des éléments mis en cache

Tous les éléments ne doivent pas être mis en cache. Le cache est pertinent pour :

- les données fréquemment consultées
- les requêtes coûteuses
- les données peu volatiles

Dans cette application, les éléments suivants ont été mis en cache :

- la liste paginée des livres
- les détails d'un livre
- le nombre total de documents

6. Mise en cache de la liste des livres avec pagination et recherche

L'un des points clés de ce TP est la mise en cache de la **page principale de l'application**, qui est la plus consultée par les utilisateurs.
Cette page combine plusieurs éléments coûteux en ressources :

- récupération paginée des livres depuis MongoDB
- calcul du nombre total de documents
- rendu Twig

Afin d'optimiser ces traitements, un cache Redis est mis en place au niveau de cette page.

6.1 Génération dynamique des clés de cache

Chaque combinaison de paramètres doit correspondre à une clé de cache unique.
Les paramètres pris en compte sont :

- le numéro de page (`page`)
- la requête de recherche (`q`) (implémenté durant le TP3 avec ElasticSearch)

```
$cacheKey = 'books_page_' . $page . '_q_' . md5($q ?? '');
```

L'utilisation de md5() permet :

- d’éviter les clés trop longues
- de garantir une clé stable même avec des chaînes complexes

Ainsi, chaque page et chaque recherche dispose de leur propre entrée Redis.

6.2 Lecture depuis Redis (cache hit)

Avant toute interaction avec MongoDB, l’application vérifie si les données sont déjà présentes dans Redis.

```
if ($redis) {
    $cached = $redis->get($cacheKey);
    if ($cached !== null) {
        $payload = json_decode($cached, true);

        echo $twig->render('index.html.twig', [
            'list'       => $payload['list'],
            'page'       => $page,
            'totalPages' => $payload['totalPages'],
            'perPage'    => $perPage,
            'total'      => $payload['total'],
            'q'          => $q,
            'cached'     => true,
        ]);
        exit;
    }
}
```

Dans ce cas :

- aucune requête MongoDB n’est exécutée.
- aucune recherche ElasticSearch n’est lancée.
- la réponse est immédiatement renvoyée à l’utilisateur.

Ce mécanisme correspond à un cache hit, garantissant des performances optimales.

6.3 Stockage du résultat dans Redis (cache miss)

Une fois les données récupérées depuis MongoDB, elles sont stockées dans Redis pour les prochaines requêtes.

```
$redis?->setex($cacheKey, 20, json_encode([
    'list'       => $list,
    'total'      => $total,
    'totalPages' => $totalPages,
]));
```

Caractéristiques du cache :

- durée de vie : 20 secondes

- format JSON
- clé dépendante de la page et de la recherche

Cette durée volontairement courte permet :

- d'assurer la fraîcheur des données
- de limiter les incohérences après modification

6.4 Indicateur de cache côté interface

Un indicateur (cached) est transmis au template Twig afin de différencier :

- une réponse issue du cache
- une réponse calculée dynamiquement

```
'cached' => true | false
```

Cela facilite :

- le débogage
- la validation du bon fonctionnement du cache

7. Cache des détails d'un livre

7.1 Clé par identifiant

```
$cacheKey = "book:" . (string) $id;
```

7.2 Lecture / écriture

```
if ($redis && $redis->exists($cacheKey)) {
    $entity = json_decode($redis->get($cacheKey), true);
} else {
    $entity = $collection->findOne(['_id' => new ObjectId($id)]);
    if ($redis && $entity) {
        $redis->setex($cacheKey, 120, json_encode($entity));
    }
}
```

Chaque livre est ainsi mis en cache individuellement.

8. Bénéfices observés

Cette stratégie de cache apporte plusieurs avantages majeurs :

- réduction drastique des accès MongoDB

- amélioration significative du temps de réponse
- meilleure résistance à la charge

Cette architecture correspond à une architecture web moderne orientée performance.

9. Conclusion du TP2

Ce TP a permis de :

- comprendre le rôle d’un cache applicatif
- intégrer Redis dans une architecture existante
- améliorer les performances de l’application
- gérer les problématiques d’invalidation
- préparer l’application à une montée en charge

Redis s’impose comme un composant essentiel dans toute architecture web moderne, notamment lorsqu’elle repose sur des bases NoSQL.