**REDIS**

**Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.**

Accès distant

Éditer /etc/redis/redis.conf

Une image contenant texte, Police, capture d’écran, typographie

Le contenu généré par l’IA peut être incorrect.

Ensuite, redémarrage du système et c’est fini pour l’installation. On peut passer à la partie 2

**Étapes 2 – Mise en place d’un cluster Redis (sharding)**

**Nettoyage de l’existant**

Arrêter le Redis de base :

**sudo systemctl stop redis-server**

**Préparer les répertoires**

Créer les dossiers pour 3 nœuds :

**mkdir -p ~/redis-cluster/7000 ~/redis-cluster/7001 ~/redis-cluster/7002**

Créer les fichiers de configuration :

**nano ~/redis-cluster/7000/redis.conf**

Contenu **:**

**port 7000**

**cluster-enabled yes**

**cluster-config-file nodes-7000.conf**

**cluster-node-timeout 5000**

**appendonly yes**

**protected-mode no**

**bind 0.0.0.0**

**Pour 7001 :** même fichier, remplacer port et cluster-config-file :

**port 7001**

**cluster-enabled yes**

**cluster-config-file nodes-7001.conf**

**cluster-node-timeout 5000**

**appendonly yes**

**protected-mode no**

**bind 0.0.0.0**

Pour 7002 : idem :

**port 7002**

**cluster-enabled yes**

**cluster-config-file nodes-7002.conf**

**cluster-node-timeout 5000**

**appendonly yes**

**protected-mode no**

**bind 0.0.0.0**

Lancer les instances

Dans 3 terminaux séparés **:**

**redis-server ~/redis-cluster/7000/redis.conf**

**redis-server ~/redis-cluster/7001/redis.conf**

**redis-server ~/redis-cluster/7002/redis.conf**

**Créer le cluster**

Quand les trois sont actifs :

**Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.redis-cli --cluster create 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 --cluster-replicas 0**

**Cluster Redis fonctionnel – sharding validé.**

**3 — Développement Application Web**

Installation d’un environnement Python

**sudo apt install python3 python3-pip**

**python3 -m venv ~/venv**

**source ~/venv/bin/activate**

**pip install flask redis-py-cluster**

**Code app.py**

**Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.**

Une image contenant texte, capture d’écran, Police, algèbre

Le contenu généré par l’IA peut être incorrect.

**4 — Démonstration**

Une image contenant texte, Police, capture d’écran, reçu

Le contenu généré par l’IA peut être incorrect.

Premier appel

Deuxième appel

Une image contenant texte, reçu, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.

Données expirables

Une image contenant texte, Police, capture d’écran, blanc

Le contenu généré par l’IA peut être incorrect.

**5-Conclusion**

**Cluster Redis configuré et opérationnel**

* L’architecture distribuée est en place : 3 nœuds reliés, slots correctement alloués.
* Chaque nœud sait gérer une partie de l’espace de clés et peut redistribuer la charge.
* Vérification par redis-cli → 16 384 slots couverts.

**Répartition automatique (slots couverts)**

* Les clés sont automatiquement réparties sur les nœuds via le sharding.
* Le cluster gère les redirections (MOVED) pour assurer la cohérence.
* Test direct : insertion sur un nœud → lecture possible sur un autre.

**Application Flask exploitant le cache Redis Cluster**

* L’application web interroge Redis avant de « simuler » une base lente.
* Connexion via RedisCluster pour prendre en compte la distribution.
* Clé stockée sur le nœud responsable, récupération automatique grâce au client cluster-aware.

**Stratégie cache-aside validée**

* Premier accès : requête lente (2 s), écriture dans Redis.
* Accès suivant : réponse immédiate → la donnée est servie depuis le cache.
* Comportement observé avec source : "db" puis source : "cache".

**Preuve d’expiration des clés**

* Chaque clé a un TTL (Time To Live) court (60 s).
* Vérifié par commande ttl foo : le cache disparaît automatiquement.
* Confirme que Redis agit bien en mémoire temporaire, non persistante.

Cette mise en place prouve :

* Le **bénéfice du cache distribué** pour soulager une base lente.
* La **simplicité d’intégration** avec une application web.
* La **robustesse** du sharding natif de Redis pour gérer charge et scalabilité.

Ce modèle cache-aside est directement transposable à une architecture réelle pour **accélérer la disponibilité des données**, tout en conservant **cohérence et performance**