Labo 4 – Load Balancing, Caching, Test de charge et Observabilité

Cours : **Architecture Logicielle (LOG430)**

Session : **Été 2025**

Date du laboratoire : **Semaine du 9 Juin 2025**

Le génie pour l'industrie

Contexte

Votre système multi-magasins a évolué vers une architecture modulaire exposée via une API RESTful (Lab 3). Ce laboratoire vise à introduire des pratiques avancées d'observabilité, de test de charge, de load balancing et d'optimisation des performances via le caching.

Objectifs d'apprentissage

- Exécuter un test de charge réaliste sur l'application.
- Observer les 4 Golden Signals (latence, trafic, erreurs, saturation).
- Ajouter des logs structurés et des métriques applicatives.
- Mettre en œuvre un répartiteur de charge.
- Implémenter un cache pour optimiser les endpoints critiques.
- Analyser et comparer les performances avant/après les optimisations.

Tâches à réaliser

Les tâches suivantes doivent être réalisées de manière **séquentielle**. L'objectif est de mesurer, à chaque étape, l'impact des améliorations apportées sur les performances globales du système. Chaque étape s'appuie sur les métriques précédemment collectées pour valider les bénéfices ou identifier les limites de la solution implémentée.

1. Test de charge initial et observabilité de base

- **Choisissez un outil de test de charge** tel que `JMeter`, `k6` ou `Artillery`. Préparez des scénarios simulant des cas d'usage réels :
  - Consultation simultanée des stocks de plusieurs magasins.
  - Génération de rapports consolidés.
  - Mise à jour de produits à forte fréquence.
- **Effectuez un test de stress** : augmentez progressivement la charge jusqu'à l'effondrement des performances ou au dépassement des limites systèmes (CPU, mémoire, erreurs).
- **Ajoutez une instrumentation minimale** pour observer le comportement de votre système :
  - Logging structuré (niveau INFO/WARN/ERROR, traçabilité des requêtes entrantes et sortantes).
  - Exposition de métriques applicatives à l'aide d'un endpoint Prometheus ou d'un module tel que Spring Actuator.
- **Configurez un système de monitoring avec Grafana** :
  - Déployez Grafana et connectez-le à Prometheus.
  - Connectez Grafana à votre source de données Prometheus.

  - Téléchargez et importez des **dashboards prédéfinis** depuis la bibliothèque officielle de Grafana (grafana.com/grafana/dashboards/) pour la visualisation des indicateurs classiques : temps de réponse, taux d'erreur, saturation CPU/mémoire, etc.
  - Créez un dashboard personnalisé si nécessaire, en vous concentrant sur les **4 Golden Signals** [1] :
    - **Latence** : temps de réponse moyen, percentiles 95e et 99e.
    - **Trafic** : nombre de requêtes par seconde.
    - **Erreurs** : taux de réponses HTTP 4xx ou 5xx.
    - **Saturation** : charge CPU, mémoire, threads, ou pool de connexions.
- **Analysez les points faibles de l'architecture** : identifiez goulets d'étranglement (pool de connexions, absence d'index, requêtes lentes) et proposez des améliorations *sans augmenter les ressources*, par exemple :
  - Ajout d'index sur les colonnes critiques.
  - Optimisation des requêtes SQL.
  - Mise en cache d'appels répétitifs.
- **Documentez vos résultats** : capturez les graphiques Grafana ou exportez les données collectées afin de créer une base de référence pour les étapes suivantes.

2. Ajout d'un Load Balancer et résilience

- **Déployez un répartiteur de charge** (ex. `NGINX`, `HAProxy`, `traefik` ou avec orchestration via Docker Compose ou Swarm).
- **Lancez N instances du service API** (N = 1, 2, 3, 4, ...) derrière le Load Balancer.
- **Répétez les tests de charge** de l'étape 1 pour chaque configuration de N instances.
- **Mesurez et comparez** :
  - Latence moyenne.
  - Requêtes par seconde.
  - Taux d'erreurs.
  - Saturation CPU/mémoire.
- **Présentez un graphique comparatif** (axe X : nombre d'instances ; axe Y : métriques de performances).
- **Test de tolérance aux pannes** : arrêtez une instance (ou VM) en cours de test et observez :
  - Continuité du service.
  - Redirection correcte par le Load Balancer.
  - Impact sur le taux d'erreurs.
- **Test de collaboration intergroupes (optionnel)** :

1. `https://sre.google/sre-book/monitoring-distributed-systems/`

- Utilisez 3 VM distinctes (1 pour le LB, 1 pour le service, 1 pour Redis/DB) pour comparer aux tests sur une VM unique.

— **Analyse des stratégies de load balancing** :
- Comparez Round Robin, Least Connections, IP Hash, Weighted Round Robin.
- Documentez les différences de comportement sous charge (cf. article de référence).

— **Documentez les limites observées** malgré la répartition de charge.

3. Implémentation d'un cache sur des endpoints critiques

— **Identifiez les endpoints les plus sollicités** ou les plus coûteux (latence élevée ou trafic important), par exemple :
- `/api/stores/{id}/stock`
- `/api/reports/sales`

— **Ajoutez un mécanisme de cache** :
- Cache mémoire local (ex. : `@Cacheable` de Spring).
- Ou cache distribué (ex. : `Redis`, `MemCache`).

— **Configurez les règles d'expiration et d'invalidation**.

— **Répétez les tests de charge** et **comparez les performances** :
- Réduction de la latence des endpoints mis en cache ?
- Diminution de la charge sur la base de données ?
- Évolution de la saturation ?

— **Consignez les gains observés** et toute limitation (ex. : données obsolètes ou risque d'incohérence).

Livrables attendus

— Code source du lab dans `lab4/`

— Scripts de test de charge

— Configuration du cache et du load balancer

— Résultats de mesure et tableaux comparatifs

— Documentation technique et README.md

— Rapport synthétique des observations et performances

Conseils

— Commencez simple, puis améliorez progressivement.

— Automatisez la collecte des métriques.

— Comparez objectivement avant/après chaque optimisation.