Comparaison des Analyses de Performance

Vue d'ensemble

Ce document compare deux analyses de performance distinctes réalisées sur l'architecture microservices :

1. **Analyse Existante** (`docs/monitoring/PERFORMANCE_ANALYSIS.md`) : Tests de montée en charge avec Nginx Load Balancer
2. **Nouvelle Analyse** (`reports/kong-load-balancer-test-report.md`) : Tests avec Kong Gateway et Load Balancer

---

1. Architecture et Configuration

1.1 Analyse Existante (Nginx + Instances Multiples)

Aspect	Configuration
Load Balancer	Nginx (manuel)
Services	API monolithique multi-instances
Instances	1 → 2 → 3 instances progressives
Base de données	Partagée entre instances
Cache	Redis (ajouté dans la dernière config)
Monitoring	Prometheus + Grafana
Gestion des pannes	Health checks Nginx

1.2 Nouvelle Analyse (Kong Gateway)

Aspect	Configuration
Load Balancer	Kong Gateway (API Gateway intégré)
Services	4 microservices distincts
Instances	3 instances inventory-api load balancées
Base de données	PostgreSQL dédiée par service
Cache	Redis par service
Monitoring	Prometheus + Grafana
Gestion des pannes	Kong upstream health checks

1.3 Comparaison Architecture

Critère	Nginx + Multi-instances	Kong + Microservices

Critère	Nginx + Multi-instances	Kong + Microservices
Complexité	Moyenne	Élevée
Séparation des responsabilités	Faible	Forte
Scalabilité	Horizontale simple	Horizontale granulaire
Point de défaillance	Nginx + DB partagée	Kong + DB par service
Gestion API	Manuelle	Centralisée (Kong)

2. Méthodologie de Test

2.1 Profils de Charge

Analyse Existante - Test de Stress Intensif

```
Durée: 5 minutes
Profil agressif:
0–30s: 0 → 100 users
30s–1min: 100 users
1–2min: 100 → 300 users
2–3min: 300 users
3–4min: 300 → 700 users
4–5min: 700 users (stress maximum)
```

Nouvelle Analyse - Test de Charge Progressive

```
Durée: 10 minutes
Profil progressif:
0–1min: 0 → 25 users
1–2min: 25 → 50 users
2–3min: 50 → 75 users
3–4min: 75 → 100 users
4–8min: 100 users (plateau)
8–10min: 100 → 0 users
```

2.2 Comparaison des Approches

Aspect	Analyse Existante	Nouvelle Analyse
Objectif	Trouver le point de rupture	Valider la stabilité
Utilisateurs max	700 (recherche de crash)	100 (validation fonctionnelle)
Escalade	Agressive et rapide	Progressive et contrôlée
Endpoints	API monolithique	4 endpoints microservices

Aspect	Analyse Existante	Nouvelle Analyse
Philosophie	Test de stress	Test de charge

3. Résultats et Métriques

3.1 Capacité Maximum

Analyse Existante

- **1 instance** : ~50 users → Crash total
- **2 instances + LB** : ~200 users → Crash Load Balancer
- **3 instances + Cache** : 300 users → Stable (seuil critique)
- **Limite identifiée** : 300 utilisateurs simultanés

Nouvelle Analyse

- **Kong + 4 microservices** : 100 users → Stable
- **Load balancing** : 3 instances inventory-api
- **Pas de crash** : Architecture résiliente
- **Limite non atteinte** : Potentiel supérieur non testé

3.2 Métriques de Performance

Métrique	Analyse Existante (300 users)	Nouvelle Analyse (100 users)
Débit	84.60 req/s	63.06 req/s
Latence moyenne	1,048ms	111.86ms
Latence p95	5,164ms	104.47ms
Taux d'erreur	12.68%	0.55%
Durée test	15 min	10 min
Total requêtes	76,319	37,953

3.3 Analyse Comparative des Performances

Performance par Utilisateur

Analyse Existante (300 users) :

- Débit par user : 84.60 ÷ 300 = 0.28 req/s/user
- Latence élevée : p95 > 5 secondes
- Erreurs significatives : 12.68%

Nouvelle Analyse (100 users) :

- Débit par user : 63.06 ÷ 100 = 0.63 req/s/user
- Latence excellente : p95 < 105ms

- Erreurs minimales : 0.55%

Conclusion Performance

La nouvelle architecture Kong offre de **meilleures performances par utilisateur** malgré un débit global inférieur, démontrant une efficacité supérieure.

---

4. Stabilité et Résilience

4.1 Gestion des Pannes

Analyse Existante

```
Problèmes identifiés:
- Worker connections limit (Nginx 1024 → 2048)
- Timeouts en cascade
- Resource exhaustion
- Crashes fréquents avec montée brutale
- Point de défaillance unique sur base de données
```

Nouvelle Analyse

```
Résilience démontrée:
- Kong Gateway stable (10 minutes sans incident)
- Load balancing efficace (3 instances inventory)
- Pas de saturation observée
- Isolation des services (microservices)
- Monitoring intégré
```

4.2 Recommandations de Capacité

Analyse Existante

```
USERS_NORMAL_OPERATION = 150     # Utilisation quotidienne
USERS_PEAK_CAPACITY = 250       # Pics de trafic
USERS_MAXIMUM_SAFE = 300        # Limite absolue
USERS_DANGER_ZONE = 300+        # Dégradation
```

Nouvelle Analyse

```
USERS_VALIDATED = 100           # Testé et stable
USERS_ESTIMATED_CAPACITY = 300+ # Potentiel estimé
```

```
USERS_RECOMMENDED_PROD = 200    # Marge de sécurité
SCALING_NEEDED = "Non testé"    # Tests supplémentaires requis
```

---

5. Qualité du Cache

5.1 Performance Cache

Analyse Existante (Redis avec 3 instances)

- **Cache hits** : 35,900 requêtes
- **Cache misses** : 22,391 requêtes
- **Hit ratio** : 61.6%
- **Impact** : Réduction significative latence

Nouvelle Analyse (Pas de test cache spécifique)

- **Cache Redis** : Configuré par service
- **Utilisation** : Non mesurée dans le test
- **Performance** : Intégrée dans les temps de réponse globaux

---

6. Monitoring et Observabilité

6.1 Métriques Collectées

Analyse Existante

- Métriques détaillées par instance
- Screenshots avant/après tests
- Analyse des crashes en temps réel
- Monitoring Grafana avec dashboards dédiés
- Health checks détaillés

Nouvelle Analyse

- Métriques k6 complètes
- Prometheus intégré
- Métriques Kong Gateway
- Pas de screenshots (Pas eu le temps de faire la configuration grafana)

---

7. Complexité et Maintenance

7.1 Complexité Opérationnelle

Analyse Existante (Nginx + Multi-instances)

Avantages :

- Configuration simple et bien documentée
- Debugging facile (moins de composants)
- Montée en charge rapide (ajout d'instances)

Inconvénients :

- Base de données partagée = point de défaillance
- Scaling monolithique moins granulaire
- Configuration manuelle du load balancing

Nouvelle Analyse (Kong + Microservices)

Avantages :

- Isolation des services (pannes limitées)
- Scaling granulaire par service
- Gestion API centralisée (Kong)
- Configuration déclarative

Inconvénients :

- Complexité d'architecture plus élevée
- Plus de composants à surveiller
- Debugging distribué plus complexe

7.2 Recommandations par Contexte

Pour Teams/Projets de Petite Taille

Recommandation : Analyse Existante (Nginx + Multi-instances)

- Configuration plus simple
- Maintenance réduite
- Performance suffisante pour la plupart des cas

Pour Systèmes d'Entreprise/Production Critique

Recommandation : Nouvelle Analyse (Kong + Microservices)

- Meilleure isolation des pannes
- Scalabilité granulaire
- Monitoring avancé
- Evolution vers cloud-native

---

8. Conclusions et Recommandations

8.1 Synthèse Comparative

Critère	Nginx + Multi-instances	Kong + Microservices	Gagnant

Critère	Nginx + Multi-instances	Kong + Microservices	Gagnant
Simplicité	⭐⭐⭐⭐⭐	⭐⭐⭐	Nginx
Performance/User	⭐⭐⭐	⭐⭐⭐⭐⭐	Kong
Stabilité	⭐⭐⭐⭐	⭐⭐⭐⭐⭐	Kong
Scalabilité	⭐⭐⭐	⭐⭐⭐⭐⭐	Kong
Maintenance	⭐⭐⭐⭐	⭐⭐⭐	Nginx
Observabilité	⭐⭐⭐⭐	⭐⭐⭐⭐⭐	Kong

8.2 Recommandations Stratégiques

Migration Progressive Recommandée

1. **Phase 1** : Démarrer avec Nginx + Multi-instances (simplicité)
2. **Phase 2** : Migration vers Kong quand les besoins augmentent
3. **Phase 3** : Microservices complets pour systèmes complexes

Seuils de Décision

- **< 200 users simultanés** : Nginx + Multi-instances suffisant
- **200-500 users** : Transition vers Kong recommandée
- **> 500 users** : Kong + Microservices obligatoire

8.3 Actions Recommandées

Court Terme

1. **Tester Kong avec charge équivalente** (300+ users)
2. **Implémenter monitoring unifié** entre les deux architectures
3. **Documenter procédures de migration**

Long Terme

1. **Stratégie cloud-native** avec Kong
2. **Auto-scaling** basé sur métriques Kong
3. **Service mesh** pour communications inter-services

Rapport de comparaison généré le : 2025-07-05
Basé sur : docs/monitoring/PERFORMANCE_ANALYSIS.md vs reports/kong-load-balancer-test-report.md