Rapport LOG430-02 – Projet BrokerX – Phase 2 (Mise à jour Phase 3)

Introduction

Cette phase inaugure la plateforme BrokerX destinée aux investisseurs particuliers. L'objectif était de livrer un prototype monolithique capable de démontrer le cycle de vie minimal d'un ordre de bourse : se connecter de manière sécurisée, disposer de fonds et placer un ordre. Les décisions architecturales ont été guidées par la recherche d'une base évolutive, testable et facilement déployable en laboratoire, tout en respectant les priorités identifiées dans le cahier des charges.
Mise à jour phase 3 : ajout d'un style événementiel (outbox + saga chorégraphiée UC-07), flux temps réel SSE pour UC-04/UC-08, comparatif de performance direct vs Gateway, et dashboards enrichis (outbox) sur Grafana (port 3001).

---

1. Analyse métier & DDD

1.1 Priorisation MoSCoW

- **Must (Phase 2 + complétée phase 3)**
  - **UC-01** Inscription & vérification d'identité (KYC/AML)
  - **UC-02** Authentification & session avec MFA
  - **UC-03** Approvisionnement du portefeuille (dépôt virtuel)
  - **UC-05** Placement d'un ordre marché/limite avec contrôles pré-trade
  - **UC-06** Modification / annulation d'un ordre actif
  - **UC-07** Appariement interne & exécution (matching moteur)
  - **UC-08** Confirmation d'exécution & notifications (ajout SSE)
- **Should** : **UC-04** abonnement temps réel aux données de marché (implémenté en SSE)
- **Could** : routage vers marchés externes simulés, sandbox de stratégies, diffusion publique de carnets
- **Won't (Phase 2/3)** : intégration réelle avec contreparties externes, smart order routing multi-places, trace distribuée inter-systèmes

1.2 UC implémentés (phase 3)

UC-04 — Abonnement aux données de marché temps réel

- **Acteurs** : Client, fournisseur de données simulé
- **Préconditions** : aucune (flux public)
- **Postcondition succès** : flux SSE `MarketTick` actif
- **Flux principal** :
  1. Le client ouvre un flux SSE : `GET /api/v1/market/stream?symbol=AAPL` (via Gateway 8081).
  2. L'adapter `MarketDataController` publie un prix simulé toutes les secondes.
  3. Le client consomme les ticks ; métriques observables via Grafana/Prometheus.
- **Exceptions** :
  - Connexion interrompue → réouverture SSE (pas de quotas avancés dans cette itération).

UC-08 — Confirmation d'exécution & notifications

- **Acteurs** : Système (notificateur), Client, Back-office
- **Préconditions** : exécution générée, canal notification configuré
- **Postcondition succès** : notification temps réel envoyée, audit append-only
- **Flux principal** :
  1. Construction d'un `ExecutionRecord` (prix, quantité, horodatage) et persistance `Notification`.
  2. Mise à jour de l'ordre et du portefeuille (positions, cash).
  3. Flux SSE : `GET /api/v1/notifications/stream` (JWT) diffuse les notifications récentes par compte ; fallback REST reste disponible.
  4. Écriture d'une trace d'audit immuable (`ExecutionAudit`).
- **Exceptions** :
  - Échec push SSE → le client peut rejouer le flux ; les notifications restent consultables via REST.

Les UC 01/02/03/05/06/07 restent identiques à la phase 2, avec l'ajout du bus événementiel/outbox pour UC-07 et la publication d'événements sur replace/cancel.

---

2. Architecture & décisions

2.1 Style hexagonal (ADR-001)

- Domaine pur : entités + invariants (onboarding, matching, notifications).
- Application : orchestrateurs qui exposent ports REST et ports événementiels.
- Adapters : Spring MVC `/api/v1/**`, documentation OpenAPI, Spring Data JPA, SSE marché/notifications.

2.6 Observabilité & Golden Signals (ADR-005)

- Micrometer + Actuator expose `/actuator/prometheus` (p95/p99) sur les UC clés.
- Métriques spécifiques : `brokerx_orders_accepted_total`, `brokerx_matching_duration_seconds`, `brokerx_matching_qty`, `brokerx_notifications_total`, `brokerx_outbox_processed_total`, `brokerx_outbox_pending`.
- Dashboards Grafana versionnés (`docs/phase2/observability`) : Golden Signals, matching/notifications, panels outbox. Grafana sur `http://localhost:3001`, Prometheus sur `http://localhost:9090`.
- Logs structurés JSON (`traceId`, `userId`, `uc`, `durationMs`, `statusCode`) via `RequestLoggingFilter`. Scripts k6 = source de charge.

2.7 API Gateway & load balancing (ADR-006)

- Gateway Kong (8081) avec LB round-robin vers `api1/api2` (monolithe) et services dédiés (`auth-service`, `portfolio-service`, `orders-service`).
- CORS maîtrisé, propagation JWT, routes SSE (marché/notifications) ouvertes.
- Health-checks agressifs + retrait auto des instances ; comparatif de perf direct (8085) vs Gateway (8081).
- Génération `X-Trace-Id` côté Gateway pour corrélation logs/métriques/audit.

2.8 Architecture événementielle (ADR-007)

- Outbox persistante (`outbox_events`) + `DomainEventBus` en mémoire ; dispatcher périodique + métriques outbox.
- Saga chorégraphiée UC-07 : `ORDER_PLACED` → matching → `ORDER_MATCHED`, publication vers notifications/audit.
- Événements sur replace/cancel (`ORDER_UPDATED`, `ORDER_CANCELED`) pour rejouer le matching et enrichir l'audit.

---

3. Technologies et stack

- **Langage/Framework** : Java 21, Spring Boot 3.2 (Web, Data JPA, Validation, Actuator), Springdoc OpenAPI
- **Base de données** : PostgreSQL 16, Redis 7
- **Observabilité** : Micrometer Prometheus, Grafana dashboards (`docs/phase2/observability`). Grafana port 3001, Prometheus port 9090.
- **Tests & performance** : JUnit 5, Mockito, Testcontainers (PostgreSQL/Redis), k6 (scénarios charge), JaCoCo.
- **Déploiement** : Docker Compose (api1/api2, gateway Kong, db, redis, prometheus, grafana), script `scripts/deploy.sh`.

---

4. Implémentation du prototype (rappel)

- Endpoints REST publics versionnés :
  - `POST /api/v1/auth/signup` (UC-01)
  - `POST /api/v1/auth/login` (UC-02)
  - `POST /api/v1/deposits` (UC-03)
  - `GET /api/v1/market/stream` (UC-04 via SSE)
  - `POST /api/v1/orders` / `PUT /api/v1/orders/{id}` / `DELETE /api/v1/orders/{id}` (UC-05/06)
  - `GET /api/v1/orders/{id}/executions` / `GET /api/v1/orders/{id}/notifications` / `GET /api/v1/notifications/stream` (UC-07/08)
  - `GET /actuator/health`, `/actuator/prometheus`

---

5. Qualité, tests & sécurité

5.1 Stratégie de tests

- **Unitaires** : onboarding, matching, notifications ; invariants domaine, règles pré-trade.
- **Intégration** : REST avec Postgres & Redis Testcontainers, idempotence, cancel/replace, matching, cache.
- **E2E** : scénarios bout-en-bout (signup → dépôt → ordre → exécution → notification).
- **Performance** : k6 (`tests/perf`) direct vs Gateway.
- **Résultats charge (5 VU, 30 s)** : Gateway (8081) p90 ~99 ms, p95 ~183 ms (~8.6 req/s, 0 % erreurs) ; Direct api1 (8085) p90 ~83 ms, p95 ~182 ms (~8.7 req/s, 0 % erreurs). Voir

`docs/phase2/observability/README.md` et Grafana (Golden Signals).
- **Couverture** : JaCoCo ; qualité de code via Spotless/Checkstyle (optionnel).

5.2 Sécurité & observabilité minimale

- MFA simulée (OTP) + verrouillage progressif (`failed_attempts`, `locked_until`)
- Journaux structurés JSON (traceId, userId, UC)
- Métriques Prometheus : latence P95/P99, RPS, taux d'erreur, saturation, outbox.
- Dashboards Grafana : vues par UC, comparatifs direct vs Gateway, panels outbox.
- Secrets : variables d'environnement (support Vault possible).

---

6. CI/CD & déploiement

- Dockerfile multi-stage, docker-compose (api, db, redis, prometheus, grafana, gateway).
- GitHub Actions : build, tests, artefacts, badge CI.
- Déploiement local : `./scripts/deploy.sh`, health via `http://localhost:8081/monolith/actuator/health`, Grafana `http://localhost:3001`.

---

7. Conclusion

La phase 3 parachève le socle en ajoutant le style événementiel (outbox + saga), les flux temps réel SSE (marché, notifications), l'enrichissement observabilité (panels outbox) et des mesures de performance comparatives Gateway vs direct. La base reste monolithique hexagonale mais prête pour l'évolution microservices/API Gateway, avec documentation, tests et métriques à l'appui.