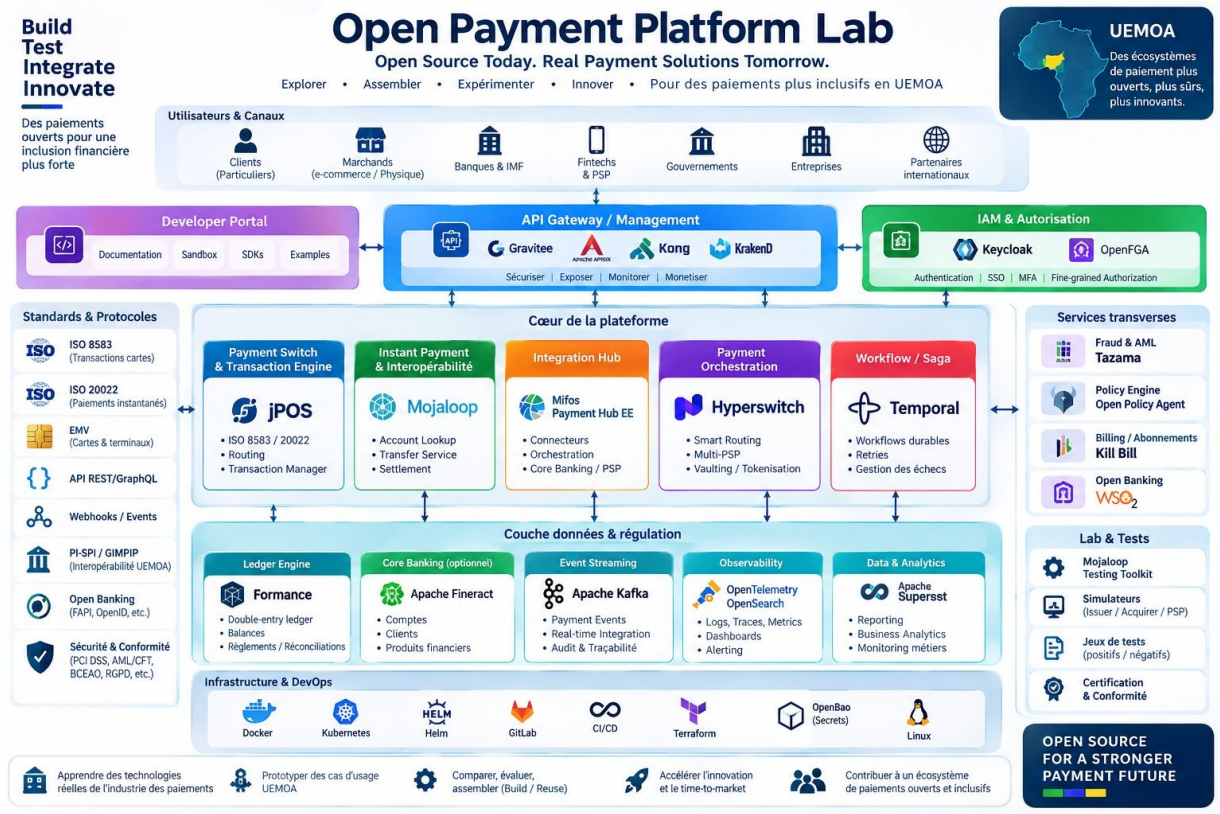
GUIDE DE MISE EN ŒUVRE · DÉPLOIEMENT · DÉVELOPPEMENT

Open Payment Platform Lab

Des composants open source. Une chaîne de paiement de bout en bout.
Pour apprendre, tester, intégrer et innover en toute souveraineté.



Document	Guide de référence — Open Payment Platform Lab
Version	1.0 — 11 septembre 2026
Auteur	Lionel SAKITI — Professionnel des infrastructures numériques
Statut	Document de travail — environnement non productif, données synthétiques
Gouvernance	Rédaction assistée par IA, gouvernée par ISO/IEC 42001 ; validation humaine requise

Same Standards · More Opportunities for Africa

Sommaire

1. Synthèse exécutive **4**
2. Vision, objectifs et périmètre **5**
2.1 Pourquoi un Lab plutôt qu'un catalogue 5
2.2 Ce que le Lab est, et ce qu'il n'est pas 5
2.3 Les huit verbes du Lab 5
2.4 Publics cibles 6
2.5 Principes directeurs 6
2.6 Périmètre 7
3. Architecture de référence **8**
3.1 Vue d'ensemble 8
3.2 Les sept couches 8
3.3 Principes d'intégration 9
3.4 Flux type d'un paiement 9
3.5 Vue étendue de la plateforme 10
3.6 Matrice des interfaces 10
4. Les dix briques prioritaires **12**
4.1 jPOS — Switch & ISO 8583 12
4.2 Prowide ISO 20022 — Messages ISO 20022 13
4.3 Mojaloop — Paiements instantanés & interopérabilité 13
4.4 Mifos Payment Hub EE — Hub d'intégration & connecteurs 14
4.5 Hyperswitch — Orchestration & routage des paiements 14
4.6 Formance — Ledger financier 15
4.7 Tazama — Fraude & transaction monitoring 16
4.8 Keycloak — IAM & authentification 16
4.9 Apache Kafka — Event streaming 17
4.10 OpenTelemetry + OpenSearch — Observabilité 17
5. Infrastructure et environnement du Lab **19**
5.1 Trois niveaux de Lab 19
5.2 Dimensionnement indicatif (niveau 2) 19
5.3 Topologie Kubernetes 20
5.4 GitOps et organisation du dépôt d'infrastructure 20
5.5 Gestion des secrets 21
5.6 Nommage, DNS et points d'accès 22
5.7 Données de test synthétiques 22
6. Déploiement pas à pas **23**
6.1 Phase 0 — Socle 23
6.2 Phase 1 — Identité, politiques et passerelle 24
6.3 Phase 2 — Événements et observabilité 25
6.4 Phase 3 — Ledger Formance 26
6.5 Phase 4 — Switch jPOS 27
6.6 Phase 5 — Adaptateur ISO 20022 27
6.7 Phase 6 — Mojaloop et Testing Toolkit 28
6.8 Phase 7 — Mifos Payment Hub EE 29
6.9 Phase 8 — Hyperswitch 29
6.10 Phase 9 — Temporal et workers de saga 30
6.11 Phase 10 — Tazama 30
6.12 Vérification globale 31
7. Développement et intégration **32**
7.1 Organisation du code applicatif 32
7.2 Modèle canonique de paiement 32
7.3 Contrat d'événements Kafka 33
7.4 Saga de paiement avec Temporal 34
7.5 Ledger : plan de comptes et Numscript 35
7.6 Adaptateur ISO 20022 avec Prowide 35
7.7 Participants jPOS 36
7.8 Workers Zeebe pour Payment Hub EE 36
7.9 Connecteurs Hyperswitch du Lab 37
7.10 Intégration Tazama 37
7.11 Politiques OPA 37
7.12 Instrumentation OpenTelemetry 38

7.13 Conventions transverses....39
8. Cas d'usage testés....40
8.1 Cas d'usage 1 — Paiement instantané wallet → wallet....40
8.2 Cas d'usage 2 — Transaction carte ISO 8583 (autorisation → annulation)....41
8.3 Cas d'usage 3 — Connexion d'un PSP via Payment Hub EE....41
8.4 Cas d'usage 4 — Simulation PI-SPI (ISO 20022)....41
8.5 Cas d'usage 5 — Scénarios de fraude avec Tazama....42
8.6 Cas d'usage 6 — Suivi bout en bout dans OpenSearch....42
8.7 Cas d'usage 7 — Tableaux de bord temps réel....43
9. Tests, simulation et benchmark....44
9.1 Pyramide de tests du Lab....44
9.2 Simulateurs....44
9.3 Jeux de tests positifs et négatifs....44
9.4 « Casser » : chaos engineering....45
9.5 Benchmark....45
10. Sécurité et conformité....47
10.1 Sécurité par défaut....47
10.2 PCI DSS : maîtriser le périmètre....47
10.3 Cadre réglementaire UEMOA et exigences transverses....47
10.4 Conformité des licences....48
11. Observabilité et exploitation....49
11.1 Les signaux surveillés....49
11.2 Alertes et runbooks....49
11.3 Sauvegarde, restauration et réinitialisation....49
12. Grille de décision Build / Buy / Reuse / Extend / Replace....50
12.1 Les cinq options....50
12.2 Critères et pondération....50
12.3 Exemple de lecture (illustratif)....50
12.4 Modèle de fiche de décision (ADR)....51
13. Gouvernance, équipe et feuille de route....52
13.1 Organisation....52
13.2 Feuille de route sur douze semaines....52
13.3 Risques et parades....52
13.4 Indicateurs de réussite du Lab....53
Annexes....54
Annexe A — Glossaire....54
Annexe B — Check-list de mise en service du Lab....54
Annexe C — Références....55

1. Synthèse exécutive

L'**Open Payment Platform Lab** (OPP Lab) est un environnement d'expérimentation qui assemble dix briques open source de référence pour reconstituer, de bout en bout, une chaîne de paiement moderne : exposition d'API, authentification, politiques métier, orchestration, switching ISO 8583, messagerie ISO 20022, paiements instantanés interopérables, ledger, détection de fraude, streaming d'événements et observabilité. Il ne s'agit pas d'un catalogue d'outils examinés isolément, mais d'une plateforme intégrée où chaque composant dialogue réellement avec les autres.

Le Lab répond à un besoin concret des équipes Platform & Software des banques, des établissements de monnaie électronique, des PSP, des switchs et des opérateurs d'infrastructures de la zone UEMOA : **comprendre les architectures de paiement par la pratique**. Installer, intégrer, tester, simuler, observer, casser, sécuriser et benchmarker sont les huit verbes qui structurent la démarche. L'ambition n'est pas d'affirmer que l'open source peut tout remplacer, mais de donner aux équipes les moyens de décider, composant par composant, en connaissance de cause : **Build, Buy, Reuse, Extend ou Replace**.

Ce guide couvre l'intégralité du cycle : architecture de référence, fiches techniques des dix briques, dimensionnement et topologie d'infrastructure, déploiement pas à pas en onze phases, développement des composants d'intégration (modèle canonique, saga, adaptateurs ISO, connecteurs, politiques, instrumentation), sept cas d'usage testés, stratégie de tests et de benchmark, sécurité et conformité, exploitation, grille de décision et gouvernance du Lab sur douze semaines.

Tableau 1 — Apports du Lab par dimension

Dimension	Ce que le Lab apporte
Apprentissage	Montée en compétence des équipes sur les standards réels de l'industrie : ISO 8583, ISO 20022, FSPIOP (Mojaloop), OAuth2/OIDC, CloudEvents, OpenTelemetry.
Architecture	Validation par l'expérience des patterns critiques : idempotence, saga et compensation, ledger à double entrée, événements immuables, corrélation des traces.
Décision	Grille objectivée Build / Buy / Reuse / Extend / Replace, alimentée par des mesures (performance, effort d'intégration, maturité, licence, compétences).
Souveraineté	Maîtrise du code, des données et des déploiements ; réduction de la dépendance aux fournisseurs ; capacité à contribuer en amont.
Écosystème UEMOA	Prototypage de cas d'usage régionaux : interopérabilité wallet ↔ banque, simulation de flux ISO 20022 de type PI-SPI, lutte contre la fraude et LBC/FT.

Positionnement du document

Le Lab est un environnement **non productif**. Il manipule exclusivement des données synthétiques et des numéros de carte de test. Toute trajectoire vers la production suppose une analyse de risques, une homologation et des décisions de licence spécifiques (voir chapitres 10 et 12).

Les captures d'écran de ce guide sont des **maquettes illustratives reconstituées** à partir des interfaces et sorties documentées des outils ; les chiffres qui y figurent sont fictifs. Elles sont destinées à être remplacées par les captures réelles de votre instance du Lab.

2. Vision, objectifs et périmètre

2.1 Pourquoi un Lab plutôt qu'un catalogue

L'industrie des paiements est réglementée, critique et exigeante. Les décisions d'architecture y engagent l'institution pour une décennie : un switch, un core banking ou un ledger ne se remplacent pas au rythme d'une application web. Dans ce contexte, évaluer un composant open source sur la base de sa documentation ou d'une démonstration commerciale ne suffit pas. Ce qui compte, c'est la manière dont il se comporte **une fois intégré** : comment il gère l'idempotence d'une requête rejouée, comment il se rétablit après la perte d'un nœud, comment il trace une transaction à travers six services, comment il se sécurise derrière une passerelle et un fournisseur d'identité.

C'est la raison d'être du Lab : faire fonctionner les outils **ensemble**. L'intégration révèle ce que l'évaluation isolée masque — frictions de modèles de données, incompatibilités de protocoles, coûts opérationnels cachés, compétences manquantes. Elle transforme une veille technologique passive en capital de compétences actif pour l'institution.

2.2 Ce que le Lab est, et ce qu'il n'est pas

Tableau 2 — Positionnement du Lab

Le Lab EST	Le Lab N'EST PAS
Un environnement d'apprentissage par la pratique pour les équipes Platform & Software	Une plateforme de production ni un produit prêt à l'emploi
Une architecture de référence intégrée, reproductible et versionnée (GitOps)	Une affirmation que l'open source doit tout remplacer
Un banc d'essai pour mesurer, comparer et benchmarker des composants	Un substitut à l'homologation réglementaire ou aux certifications (PCI DSS, schémas de cartes)
Un support de décision Build / Buy / Reuse / Extend / Replace	Un environnement traitant des données clients réelles ou des PAN réels
Un outil de dialogue entre DSI, métiers, risques, conformité et régulateur	Un projet figé : la liste des briques évolue avec les retours d'expérience

2.3 Les huit verbes du Lab

Tableau 3 — Les huit verbes et leurs livrables

Verbe	Objectif opérationnel	Livrable attendu
Installer	Déployer chaque brique de manière reproductible, sans intervention manuelle	Charts Helm / manifestes versionnés, runbook d'installation
Intégrer	Faire dialoguer les briques via des contrats explicites (API, événements)	Contrats OpenAPI / AsyncAPI, adaptateurs, tests de contrat
Tester	Vérifier le comportement fonctionnel nominal et les cas d'erreur	Suites de tests automatisées, jeux de données positifs / négatifs
Simuler	Reproduire des acteurs absents : émetteurs, acquéreurs, DFSP, PSP, régulateur	Simulateurs et bouchons (Testing Toolkit, mocks jPOS, dummy connectors)

Verbe	Objectif opérationnel	Livrable attendu
Observer	Voir chaque transaction à travers tous les services	Traces distribuées, métriques, journaux corrélés, tableaux de bord
Casser	Provoquer des pannes pour valider la résilience	Scénarios de chaos, rapports de reprise, RTO / RPO mesurés
Sécuriser	Appliquer sécurité par défaut et contrôles de conformité	Politiques, secrets, mTLS, SBOM, rapports de scan
Benchmarker	Mesurer débit, latence, coût de ressources	Rapports de performance comparables et reproductibles

2.4 Publics cibles

- **Équipes Platform Engineering et SRE** : socle Kubernetes, GitOps, observabilité, résilience, sécurité de la chaîne d'approvisionnement logicielle.
- **Équipes Software / Intégration** : développement des adaptateurs, connecteurs, workflows et politiques ; maîtrise des standards ISO 8583 et ISO 20022.
- **Architectes d'entreprise et de solutions** : validation des patterns, élaboration des feuilles de route, arbitrages Build / Buy.
- **Risques, conformité, sécurité et LBC/FT** : compréhension concrète des contrôles, de la traçabilité et des mécanismes de détection.
- **Directions métiers et directions générales** : démonstrateurs tangibles pour éclairer les décisions d'investissement et de partenariat.
- **Écoles, universités et communautés techniques** : support pédagogique aligné sur les besoins réels de l'industrie régionale.

2.5 Principes directeurs

Tableau 4 — Principes directeurs

Principe	Traduction concrète dans le Lab
Souveraineté par la maîtrise	Code source accessible, déploiement sur infrastructure maîtrisée (on-premise ou cloud souverain), aucune dépendance SaaS obligatoire.
Standards ouverts d'abord	ISO 8583, ISO 20022, FSPIOP, OAuth2 / OIDC, OpenAPI, AsyncAPI, CloudEvents, OpenTelemetry, BPMN.
Reproductibilité	Tout est code : infrastructure, configuration, politiques, jeux de données. Un Lab doit pouvoir être reconstruit en moins d'une journée.
Sécurité par défaut	Deny-all réseau, TLS partout, secrets hors dépôt, moindre privilège, images scannées et signées.
Données synthétiques uniquement	Aucun PAN réel, aucune donnée client réelle ; générateurs de données reproductibles et documentés.
Observabilité native	Chaque service émet traces, métriques et journaux corrélés par un identifiant de transaction et un traceparent W3C.
Décisions documentées	Chaque arbitrage fait l'objet d'un ADR (Architecture Decision Record) daté, argumenté et mesuré.

2.6 Périmètre

Dans le périmètre : les dix briques prioritaires et leurs intégrations ; les composants de socle nécessaires (Kubernetes, GitOps, secrets, registre, bases de données) ; les sept cas d'usage de référence ; les simulateurs d'acteurs externes ; la chaîne de tests, de benchmark et d'observabilité.

Hors périmètre de la version 1.0 : la connexion à des systèmes réels (schémas de cartes, PI-SPI, STAR-UEMOA, core banking de production) ; la certification PCI DSS ; les modules HSM physiques (remplacés par un HSM logiciel de test) ; la haute disponibilité multi-sites. Ces éléments sont traités comme des extensions de la feuille de route.

3. Architecture de référence

3.1 Vue d'ensemble

L'architecture du Lab s'organise autour d'une chaîne verticale — des canaux jusqu'à l'observabilité — et de services transverses (identité, politiques, fraude, workflow). La figure ci-dessous synthétise la cible autour des dix outils prioritaires.

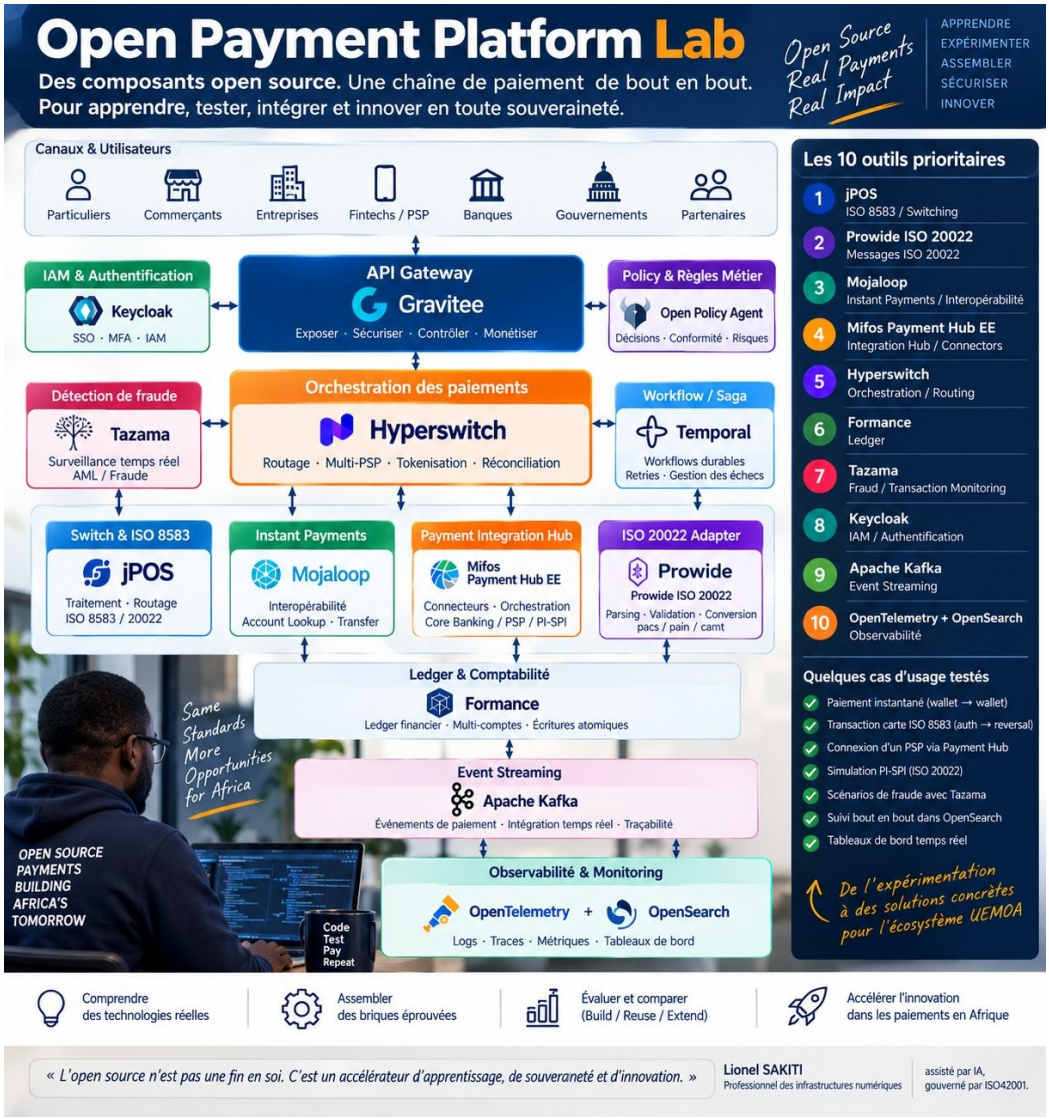


Figure 1 — Architecture de référence du Open Payment Platform Lab — 10 briques prioritaires

3.2 Les sept couches

Tableau 5 — Les sept couches de l'architecture

Couche	Rôle	Briques du Lab
1. Canaux & utilisateurs	Points d'entrée : particuliers, commerçants, entreprises, fintechs, banques, administrations, partenaires	Applications de démonstration, simulateurs
2. Exposition & sécurité	Exposer, sécuriser, contrôler et monétiser les API ; authentifier ; appliquer les règles métier	Gravitee, Keycloak, Open Policy Agent

Couche	Rôle	Briques du Lab
3. Orchestration	Router les paiements, gérer le multi-PSP, la tokenisation, la réconciliation ; garantir la durabilité des workflows	Hyperswitch, Temporal
4. Rails & protocoles	Traiter les flux cartes, instantanés, intégrations et messages ISO 20022	jPOS, Mojaloop, Mifos Payment Hub EE, Prowide ISO 20022
5. Ledger & comptabilité	Source de vérité financière : écritures atomiques, multi-comptes, double entrée	Formance
6. Événements	Diffuser les événements de paiement en temps réel ; assurer traçabilité et découplage	Apache Kafka
7. Observabilité & risque	Logs, traces, métriques, tableaux de bord ; surveillance fraude / AML en temps réel	OpenTelemetry, OpenSearch, Tazama

3.3 Principes d'intégration

L'intégration des briques obéit à six règles non négociables, qui constituent en elles-mêmes l'un des principaux enseignements du Lab :

- **API d'abord, contrats explicites.** Chaque interface synchrone est décrite en OpenAPI 3 ; chaque flux asynchrone en AsyncAPI. Les contrats sont versionnés et testés (tests de contrat) avant toute intégration.
- **Idempotence de bout en bout.** Toute requête de paiement porte une clé d'idempotence (`Idempotency-Key`) propagée jusqu'au ledger. Un rejeu ne doit jamais produire une double écriture.
- **Saga plutôt que transaction distribuée.** Aucune transaction XA n'est tentée entre services. Les étapes sont orchestrées par un workflow durable (Temporal) qui exécute des compensations explicites en cas d'échec.
- **Le ledger est la source de vérité.** Les soldes ne sont jamais calculés ailleurs que dans Formance. Les autres systèmes consomment des projections.
- **Événements immuables.** Chaque changement d'état significatif est publié sur Kafka au format CloudEvents ; les événements ne sont jamais modifiés, seulement complétés.
- **Corrélation universelle.** Un identifiant de paiement métier (`paymentId`) et un contexte de trace W3C (`traceparent`) accompagnent chaque appel, chaque message et chaque événement.

3.4 Flux type d'un paiement

Le flux nominal d'un paiement initié par API illustre la collaboration entre les couches. Il est détaillé au chapitre 8 (cas d'usage 1) et se résume ainsi :

1. Le canal appelle `POST /payments` sur Gravitee avec un jeton OAuth2 émis par Keycloak et une clé d'idempotence.
2. Gravitee valide le jeton, applique quotas et limites de débit, puis consulte OPA pour les règles d'accès contextuelles.
3. Hyperswitch reçoit le paiement, applique ses règles de routage et démarre un workflow `PaymentSagaWorkflow` dans Temporal.
4. Le workflow vérifie les politiques métier (OPA), soumet la transaction à Tazama pour évaluation fraude, puis bloque les fonds dans Formance.
5. Le paiement est émis sur le rail choisi : Mojaloop (instantané), jPOS (carte ISO 8583), Payment Hub EE (connecteur PSP / core banking) ou adaptateur ISO 20022.

6. À la confirmation, le workflow valide l'écriture définitive dans Formance ; en cas d'échec, il libère les fonds (compensation).
7. Chaque étape publie un événement sur Kafka et émet des spans OpenTelemetry, agrégés dans OpenSearch pour le suivi bout en bout.

3.5 Vue étendue de la plateforme

Au-delà des dix briques prioritaires, l'architecture cible étendue identifie des composants complémentaires à expérimenter dans un second temps : passerelles alternatives (Kong, Apache APISIX, KrakenD), autorisation fine (OpenFGA), facturation et abonnements (Kill Bill), open banking (WSO2), core banking optionnel (Apache Fineract), analytique (Apache Superset), ainsi que le socle Infrastructure & DevOps (Docker, Kubernetes, Helm, GitLab, CI/CD, Terraform, OpenBao, Linux). Elle intègre également un portail développeur (documentation, sandbox, SDK) et un pôle Lab & Tests (Mojaloop Testing Toolkit, simulateurs émetteur / acquéreur / PSP, jeux de tests, certification).

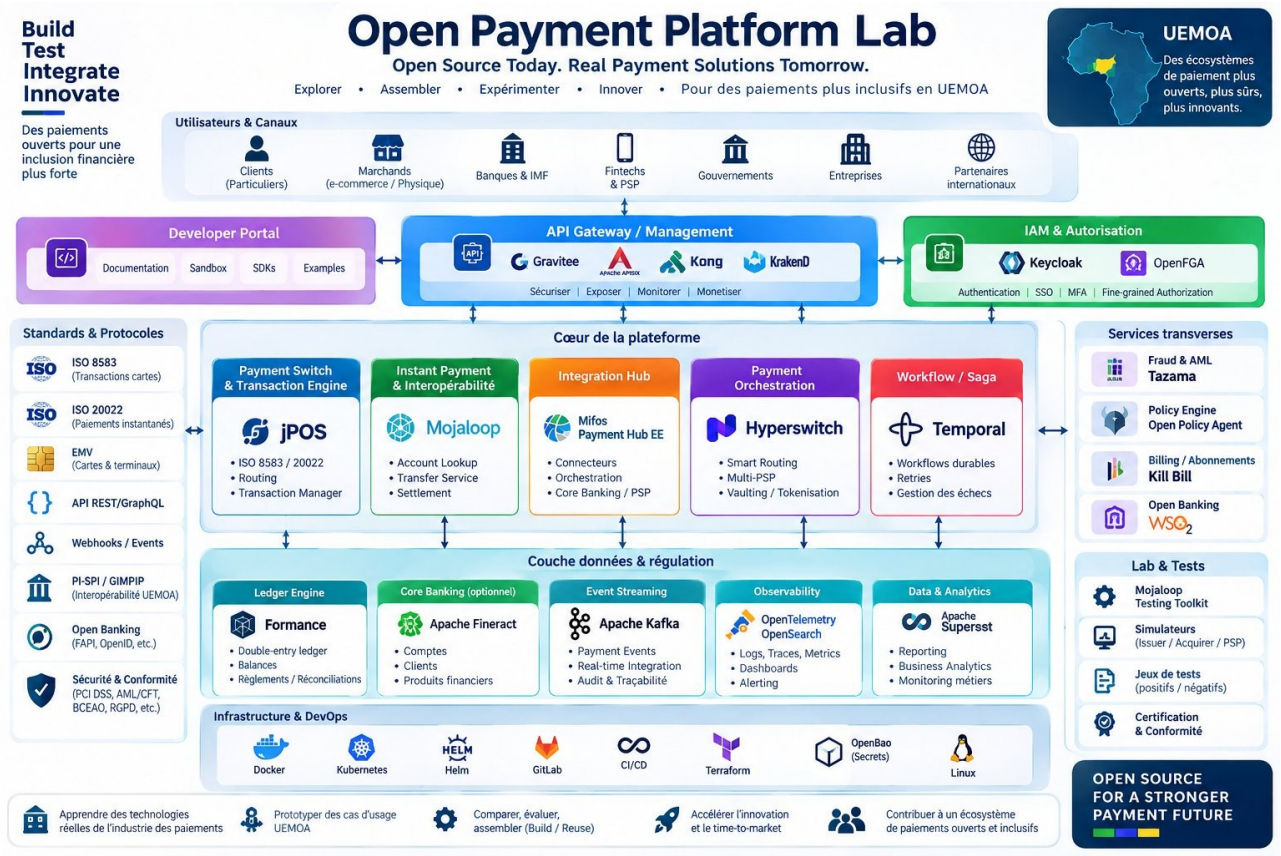


Figure 2 — Vue étendue — cœur de plateforme, couche données & régulation, services transverses

3.6 Matrice des interfaces

Tableau 6 — Matrice des interfaces du Lab

De → Vers	Protocole	Format / contrat	Sécurité
Canal → Gravitee	HTTPS REST	OpenAPI « Payments API v1 »	OAuth2 / OIDC (Keycloak), PKCE, mTLS partenaires
Gravitee → OPA	HTTP (callout)	JSON input / décision	Réseau interne, NetworkPolicy
Gravitee → Hyperswitch	HTTPS REST	API Payments Hyperswitch	Clé API marchand injectée par la passerelle

De → Vers	Protocole	Format / contrat	Sécurité
Hyperswitch → Temporal	gRPC	SDK Temporal (connecteur personnalisé)	mTLS interne
Worker → Tazama	HTTP REST	ISO 20022 JSON (pacs.008, pacs.002, pain.001, pain.013)	Réseau interne
Worker → Formance	HTTP REST	API Ledger v2 + Numscript	Jeton de service (client credentials)
Worker → Mojaloop	HTTP REST	FSPIOP (parties, quotes, transfers)	mTLS, signature JWS selon configuration
Terminal / acquéreur → jPOS	TCP	ISO 8583 (packager générique)	TLS, liste blanche IP
Payment Hub EE ↔ systèmes	HTTP / Kafka	Connecteurs (AMS, Mojaloop, GSMA, canaux)	Selon connecteur
Adaptateur ISO 20022 ↔ rails	HTTPS / MQ simulé	XML ISO 20022 (pacs, pain, camt)	mTLS, validation XSD
Tous → Kafka	Kafka protocol	CloudEvents JSON (schémas versionnés)	SASL/SCRAM + TLS, ACL par topic
Tous → OTel Collector	OTLP gRPC / HTTP	Traces, métriques, logs	Réseau interne

4. Les dix briques prioritaires

Ce chapitre présente chaque brique sous forme de fiche standardisée : rôle, caractéristiques techniques, place dans le Lab et points d'attention. Les versions ne sont volontairement pas figées : le Lab épingle les versions dans son dépôt GitOps et les met à jour de manière contrôlée. Les informations de licence doivent être vérifiées sur le dépôt officiel au moment de l'installation.

Tableau 7 — Synthèse des dix briques (licences à confirmer sur les dépôts officiels)

Brique	Licence	Langage	Couche	Priorité Lab
jPOS	AGPL v3 / commerciale	Java	Rails	Vague 2
Prowide ISO 20022	Apache 2.0	Java	Rails	Vague 2
Mojaloop	Apache 2.0	Node.js	Rails	Vague 3
Mifos Payment Hub EE	MPL 2.0 + Zeebe	Java	Rails / intégration	Vague 3
Hyperswitch	Apache 2.0	Rust	Orchestration	Vague 3
Formance	MIT	Go	Ledger	Vague 2
Tazama	Apache 2.0	TypeScript	Risque	Vague 4
Keycloak	Apache 2.0	Java	Sécurité	Vague 1
Apache Kafka	Apache 2.0	Java / Scala	Événements	Vague 1
OpenTelemetry + OpenSearch	Apache 2.0	Go / Java	Observabilité	Vague 1

4.1 jPOS — Switch & ISO 8583

Caractéristique	Description
Fonction	Framework Java de référence pour le traitement des messages ISO 8583 : packagers, canaux, serveurs, gestionnaire de transactions (TransactionManager)
Licence	GNU AGPL v3 ; licence commerciale disponible auprès de l'éditeur
Technologie	Java, conteneur Q2 (déploiement par descripteurs XML), jPOS-EE pour les modules étendus
Déploiement Lab	Image conteneur construite depuis jPOS-template (Gradle), StatefulSet dans ns/opp-switch
Interfaces	TCP ISO 8583 (entrée terminaux / acquéreurs simulés), HTTP sortant vers OPA, Tazama, Formance ; Kafka

Rôle dans le Lab

jPOS joue le rôle de switch monétique : il reçoit des demandes d'autorisation (MTI 0100/0200), des annulations (0400) et des messages réseau (0800), les fait transiter par une chaîne de participants — contrôle de champs, politique, fraude, réservation de fonds — et répond (0110/0210/0410). Il est le banc idéal pour comprendre la mécanique ISO 8583, les codes réponse, la gestion des timeouts et des reversals.

Points d'attention

- La licence AGPL impose de publier les modifications si le logiciel est mis à disposition via un réseau : l'usage en production doit faire l'objet d'une analyse juridique ou d'une licence commerciale.

- Chaque réseau et chaque acquéreur utilise sa propre variante ISO 8583 (version 1987/1993, champs privés) : le packager doit être configuré en conséquence.
- Le Lab n'utilise que des PAN de test et un HSM logiciel ; aucune clé réelle ne doit y transiter.

4.2 Prowide ISO 20022 — Messages ISO 20022

Caractéristique	Description
Fonction	Bibliothèque Java pour construire, analyser, convertir et manipuler les messages ISO 20022 (pacs, pain, camt…) avec un modèle objet complet
Licence	Apache 2.0 pour la bibliothèque open source ; offre commerciale (Prowide Integrator) pour les fonctions avancées de validation et de traduction MT ↔ MX
Technologie	Java, artefacts Maven, classes générées depuis les schémas ISO 20022
Déploiement Lab	Intégrée dans le microservice iso20022-adapter (Spring Boot), ns/opp-iso20022
Interfaces	REST / XML en entrée, événements Kafka en sortie, appels vers Tazama et le workflow

Rôle dans le Lab

Prowide sert de fondation à l'**ISO 20022 Adapter** du Lab : réception d'un pacs.008 ou d'un pain.001, validation, conversion vers le modèle canonique interne, production des réponses pacs.002 et des notifications camt.054. C'est aussi le composant utilisé pour simuler des échanges de type PI-SPI, la plateforme de paiement instantané interopérable de la BCEAO qui s'appuie sur ISO 20022.

Points d'attention

- La validation de règles de marché (au-delà du XSD) est à construire ou à acquérir : c'est un cas d'école pour la grille Build / Buy.
- Les versions de messages évoluent (pacs.008.001.08, .10…) : le Lab doit épingler les versions et tester la compatibilité.
- Les spécifications détaillées d'un système réel (usages des champs, codes motifs) ne sont pas publiques : le Lab travaille sur des guides d'implémentation génériques.

4.3 Mojaloop — Paiements instantanés & interopérabilité

Caractéristique	Description
Fonction	Hub de paiement instantané interopérable : découverte de compte (account lookup), devis (quotes), transferts, règlement (settlement)
Licence	Apache 2.0 ; projet soutenu par la Fondation Mojaloop
Technologie	Microservices Node.js, MySQL, Kafka, MongoDB, Redis ; déploiement par charts Helm
Déploiement Lab	Charts Helm Mojaloop dans ns/opp-mojaloop, avec Testing Toolkit (TTK) et DFSP simulés
Interfaces	API FSPIOP (REST asynchrone à callbacks) ; API d'administration ; support ISO 20022 dans les versions récentes

Rôle dans le Lab

Mojaloop incarne dans le Lab le **rail de paiement instantané interopérable** entre fournisseurs de services financiers (DFSP) : wallets, banques, IMF. Il permet d'expérimenter la résolution d'alias (numéro de téléphone),

le modèle en trois phases (lookup, quote, transfer), les limites de position nette (NDC) et les fenêtres de règlement — autant de concepts directement transposables aux systèmes de paiement instantané régionaux.

Points d'attention

- Le déploiement complet est exigeant en ressources : prévoir un nœud dédié ou utiliser un profil allégé pour le niveau 1.
- L'API FSPIOP est asynchrone (callbacks PUT) : les connecteurs doivent gérer la corrélation et les timeouts.
- Mojaloop n'est pas PI-SPI : il sert de support pédagogique et de simulateur, pas de réplique d'un système opéré par la Banque Centrale.

4.4 Mifos Payment Hub EE — Hub d'intégration & connecteurs

Caractéristique	Description
Fonction	Couche d'orchestration et d'intégration entre canaux, systèmes de gestion de comptes (core banking) et systèmes de paiement externes
Licence	Mozilla Public License 2.0 pour les dépôts vérifiés ; moteur de workflow Zeebe (Camunda) sous licence propre
Technologie	Java / Spring Boot, workflows BPMN exécutés par Zeebe, Kafka, Elasticsearch / OpenSearch, charts Helm
Déploiement Lab	ns/opp-phee : moteur Zeebe, connecteurs (Mojaloop, AMS, GSMA, canal), application Operations
Interfaces	API canal, connecteurs AMS (Fineract ou autre), connecteur Mojaloop, API GSMA Mobile Money, flux bulk

Rôle dans le Lab

Payment Hub EE représente dans le Lab le **côté participant** : il permet à un DFSP simulé de se connecter à Mojaloop, d'orchestrer ses flux via des diagrammes BPMN et d'interfacer son core banking (Apache Fineract en option). Il est particulièrement utile pour les paiements de masse (G2P, salaires) et les intégrations mobile money au standard GSMA.

Points d'attention

- **Licence du moteur Zeebe** : depuis la version 8.6, Camunda exige une licence pour l'usage en production des composants auto-hébergés ; les versions 8.5 et antérieures restent utilisables sans licence commerciale. Le Lab, non productif, n'est pas concerné, mais toute trajectoire vers la production doit trancher ce point.
- Plusieurs dépôts composent la solution : vérifier la licence et la maturité de chaque connecteur utilisé.
- La valeur réside dans les connecteurs : prévoir l'effort de développement pour les systèmes propres à l'institution.

4.5 Hyperswitch — Orchestration & routage des paiements

Caractéristique	Description
Fonction	Orchestrateur de paiements : routage intelligent multi-PSP, retries, tokenisation / coffre-fort de cartes, réconciliation, tableau de bord
Licence	Apache 2.0 (Juspay)

Caractéristique	Description
Technologie	Rust (router), PostgreSQL, Redis ; Control Center (interface web) ; SDK web
Déploiement Lab	ns/opp-orchestr : router, scheduler, Control Center ; connecteurs de test et connecteurs personnalisés
Interfaces	API REST Payments / Refunds, webhooks, connecteurs vers PSP ; connecteurs Lab vers le workflow et les rails

Rôle dans le Lab

Hyperswitch est la **tour de contrôle** des paiements marchands : il choisit le meilleur chemin pour chaque transaction selon des règles (montant, moyen de paiement, pays), bascule vers un PSP de secours en cas d'échec et centralise l'analyse des taux d'autorisation. Dans le Lab, il est également le point de départ des sagas Temporal pour les rails internes.

Points d'attention

- Les connecteurs vers les rails africains (mobile money, systèmes nationaux) sont à développer : c'est l'une des principales opportunités de contribution en amont.
- Le périmètre PCI DSS s'étend dès que le coffre-fort de cartes est activé : le Lab l'utilise uniquement avec des cartes de test.
- Bien distinguer l'orchestration commerciale (Hyperswitch) de l'orchestration de processus durable (Temporal).

4.6 Formance — Ledger financier

Caractéristique	Description
Fonction	Ledger programmable à double entrée : comptes, écritures atomiques multi-postings, soldes, historique immuable
Licence	MIT pour le Formance Ledger
Technologie	Go, PostgreSQL ; langage Numscript pour décrire les mouvements de fonds ; API v2
Déploiement Lab	ns/opp-ledger : service ledger + PostgreSQL (StatefulSet), outil en ligne de commande
Interfaces	API REST (transactions, comptes, volumes), idempotence par clé, métadonnées

Rôle dans le Lab

Formance est la **source de vérité financière** du Lab. Chaque paiement se traduit par des écritures atomiques et équilibrées : réservation (hold), règlement, frais, commissions, remboursements. Numscript rend ces mouvements lisibles par les équipes métier et comptables, ce qui en fait un formidable outil pédagogique sur la comptabilité des paiements.

Points d'attention

- Définir le plan de comptes avant tout développement : c'est une décision métier, pas technique.
- Gérer la précision monétaire : le franc CFA (XOF) n'a pas de subdivision, mais les frais en pourcentage exigent des règles d'arrondi explicites.
- Le ledger n'est pas un core banking : il ne gère ni produits, ni clients, ni intérêts.

4.7 Tazama — Fraude & transaction monitoring

Caractéristique	Description
Fonction	Surveillance transactionnelle en temps réel : règles, typologies, décision et alertes, sur des messages ISO 20022
Licence	Apache 2.0 ; projet de la Linux Foundation
Technologie	Microservices (TMS API, event director, processeurs de règles, processeur de typologies, TADProc), messagerie NATS, PostgreSQL, Valkey
Déploiement Lab	ns/opp-fraud : pile Docker Compose (niveau 1) ou manifestes Kubernetes (niveau 2)
Interfaces	API d'évaluation (pacs.008, pacs.002, pain.001, pain.013) ; relais de résultats vers NATS, REST, Kafka

Rôle dans le Lab

Tazama apporte au Lab une capacité de **détection de fraude et de surveillance LBC/FT en temps réel**, conçue dès l'origine pour les systèmes de paiement instantané et les marchés émergents. Chaque transaction traverse des règles (vélocité, montants anormaux, nouveaux bénéficiaires…) agrégées en typologies, avec des seuils d'alerte et d'interdiction.

Points d'attention

- La pile de référence publiée est destinée au développement et aux bacs à sable, pas à la production.
- La qualité de détection dépend des règles et du calibrage : prévoir un travail conjoint avec la conformité.
- L'interdiction en ligne ajoute de la latence : mesurer son impact sur les parcours instantanés.

4.8 Keycloak — IAM & authentification

Caractéristique	Description
Fonction	Gestion des identités et des accès : SSO, OIDC / OAuth2, SAML, MFA, fédération, comptes de service
Licence	Apache 2.0 ; projet de la CNCF
Technologie	Java (Quarkus), PostgreSQL ; realms, clients, rôles, politiques client (profils de type FAPI)
Déploiement Lab	ns/opp-iam : opérateur Keycloak ou chart Helm, realm opp-lab importé depuis Git
Interfaces	Endpoints OIDC (authorize, token, introspect, JWKS), API d'administration

Rôle dans le Lab

Keycloak émet et valide toutes les identités du Lab : utilisateurs des applications de démonstration, partenaires PSP (client credentials avec mTLS), services internes (comptes de service des workers) et opérateurs (accès aux consoles). Il permet d'expérimenter les exigences d'authentification forte et les profils de sécurité de type open banking.

Points d'attention

- Le realm doit être géré en code (export / import versionné) : aucune configuration manuelle durable.
- La durée de vie des jetons et la rotation des clés sont des paramètres de sécurité à documenter.
- Séparer strictement le realm d'administration (master) du realm applicatif.

4.9 Apache Kafka — Event streaming

Caractéristique	Description
Fonction	Plateforme distribuée de streaming d'événements : topics partitionnés, rétention, rejouabilité
Licence	Apache 2.0
Technologie	Java / Scala ; mode KRaft (sans ZooKeeper) ; opérateur Strimzi sur Kubernetes
Déploiement Lab	ns/opp-streaming : cluster 3 brokers KRaft, topics et utilisateurs déclarés en CRD
Interfaces	Protocole Kafka (TLS + SASL), schémas d'événements CloudEvents versionnés

Rôle dans le Lab

Kafka est le **système nerveux** du Lab : chaque étape significative d'un paiement y est publiée. Il découple les producteurs des consommateurs (analytique, fraude, notifications, audit), permet de rejouer l'historique et constitue la piste d'audit technique. Plusieurs briques (Mojaloop, Payment Hub EE) l'utilisent aussi en interne.

Points d'attention

- Définir la clé de partition (paymentId ou compte) pour garantir l'ordre des événements d'un même paiement.
- Les événements contiennent des données personnelles : chiffrement, rétention et droit d'accès à encadrer.
- La sémantique « exactly-once » a un coût : le Lab privilégie at-least-once avec consommateurs idempotents.

4.10 OpenTelemetry + OpenSearch — Observabilité

Caractéristique	Description
Fonction	OpenTelemetry : standard de collecte des traces, métriques et journaux. OpenSearch : stockage, recherche, tableaux de bord, analyse de traces, alerting
Licence	Apache 2.0 (OpenTelemetry, projet CNCF ; OpenSearch, Fondation OpenSearch / Linux Foundation)
Technologie	Agents et SDK OTel, OTel Collector, Data Prepper, cluster OpenSearch, OpenSearch Dashboards
Déploiement Lab	ns/opp-observ : Collector (DaemonSet + Deployment), OpenSearch 3 nœuds, Dashboards
Interfaces	OTLP gRPC 4317 / HTTP 4318, API OpenSearch 9200, Dashboards 5601

Rôle dans le Lab

Le couple OpenTelemetry + OpenSearch permet de **voir** chaque transaction traverser la plateforme : trace distribuée du canal jusqu'au ledger, métriques de performance et d'erreur, journaux corrélés, tableaux de bord métier. C'est la condition de tous les autres verbes du Lab : on ne peut ni tester, ni casser, ni benchmarker ce qu'on n'observe pas.

Points d'attention

- Masquer systématiquement les données sensibles (PAN, identifiants) dans les attributs de spans et les journaux.

- Maîtriser l'échantillonnage : 100 % dans le Lab, stratégie tail-based en contexte de charge.
- Dimensionner la rétention : les traces consomment rapidement le stockage.

5. Infrastructure et environnement du Lab

5.1 Trois niveaux de Lab

Le Lab est conçu pour être accessible à des organisations de tailles très différentes. Trois niveaux sont proposés ; ils partagent le même dépôt et les mêmes contrats, seule la topologie change. Il est recommandé de commencer au niveau 1 pour la formation individuelle et de passer au niveau 2 dès que l'équipe travaille sur les intégrations.

Tableau 8 — Les trois niveaux de Lab

Niveau	Cible	Socle	Briques	Usage
1 — Poste développeur	1 personne	Docker / Podman Compose, 1 machine (≥ 32 Go RAM)	Sous-ensembles par cas d'usage (3 à 5 briques simultanées)	Découverte, formation, développement d'un adaptateur
2 — Lab d'équipe	5 à 20 personnes	Kubernetes léger (k3s ou RKE2), 3 nœuds	Les 10 briques + socle	Intégration complète, cas d'usage bout en bout, démonstrations
3 — Lab de performance	Équipe plateforme	Kubernetes multi-nœuds, stockage distribué, nœuds dédiés	Les 10 briques en configuration HA	Benchmark, chaos engineering, préfiguration d'architecture

5.2 Dimensionnement indicatif (niveau 2)

Les valeurs ci-dessous sont des ordres de grandeur pour un Lab d'équipe à charge modérée ; elles doivent être ajustées après les premières mesures. Elles incluent les dépendances (bases de données, caches) propres à chaque brique.

Tableau 9 — Dimensionnement indicatif du Lab de niveau 2

Domaine	vCPU (requests)	RAM	Stockage	Remarques
Socle (Argo CD, ingress, cert-manager, OpenBao, registre)	4	8 Go	100 Go	Registre d'images local recommandé
IAM & passerelle (Keycloak, OPA, Gravitee + base)	4	8 Go	20 Go	Gravitee avec base de configuration et analytics
Kafka (3 brokers KRaft)	3	6 Go	150 Go	Stockage rapide (SSD)
Observabilité (OTel, OpenSearch 3 nœuds, Dashboards)	6	16 Go	200 Go	Poste le plus consommateur en stockage
Formance + PostgreSQL	2	4 Go	50 Go	Sauvegardes quotidiennes
jPOS + adaptateur ISO 20022	2	4 Go	10 Go	JVM à calibrer
Mojaloop + dépendances + TTK	8	20 Go	60 Go	Brique la plus lourde du Lab
Payment Hub EE (Zeebe, connecteurs, Operations)	6	12 Go	40 Go	Zeebe exige des disques performants
Hyperswitch (router, Control Center, PostgreSQL, Redis)	3	6 Go	20 Go	
Temporal (+ base) et workers du Lab	3	6 Go	20 Go	

Domaine	vCPU (requests)	RAM	Stockage	Remarques
Tazama (processeurs, NATS, PostgreSQL, Valkey)	4	8 Go	20 Go	Nombre de processeurs de règles variable
Total indicatif	≈ 45	≈ 98 Go	≈ 690 Go	Soit 3 nœuds de 16 vCPU / 64 Go avec marge

5.3 Topologie Kubernetes

Chaque domaine fonctionnel dispose de son propre namespace, ce qui permet d'appliquer des quotas de ressources, des politiques réseau et des droits d'accès différenciés. La règle réseau par défaut est « tout interdire » ; chaque flux autorisé est déclaré explicitement dans une NetworkPolicy versionnée, en cohérence avec la matrice des interfaces (§ 3.6).

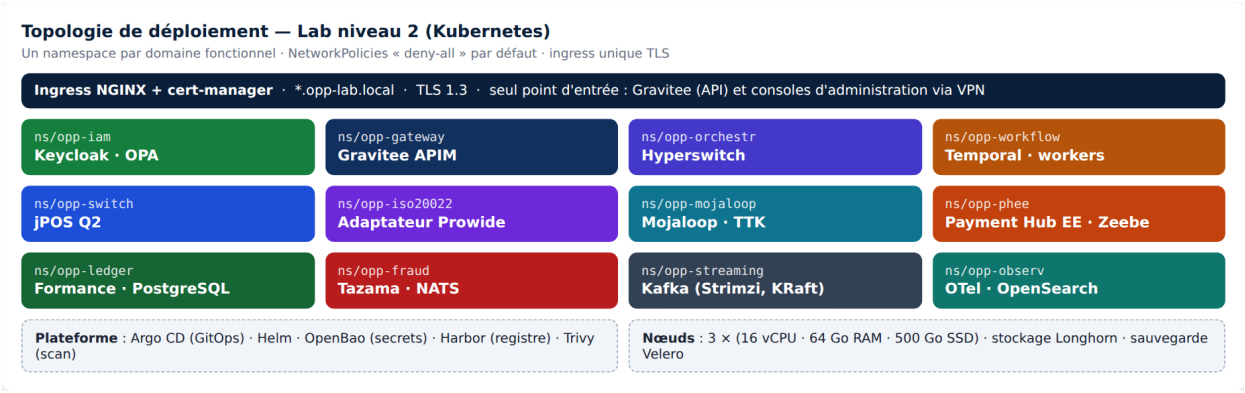


Figure 3 — Topologie du Lab de niveau 2 — un namespace par domaine

platform/policies/network/opp-ledger.yaml — seul le worker de saga peut écrire dans le ledger

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-saga-worker-to-ledger
  namespace: opp-ledger
spec:
  podSelector:
    matchLabels: { app.kubernetes.io/name: formance-ledger }
  policyTypes: [Ingress]
  ingress:
    - from:
        - namespaceSelector:
            matchLabels: { kubernetes.io/metadata.name: opp-workflow }
          podSelector:
            matchLabels: { app: payment-saga-worker }
      ports:
        - { protocol: TCP, port: 8080 }
```

5.4 GitOps et organisation du dépôt d'infrastructure

L'état complet du Lab est décrit dans un dépôt Git unique (`opp-lab-gitops`), synchronisé en continu par Argo CD. Toute modification passe par une merge request revue ; aucune commande `kubectl apply` manuelle n'est tolérée au-delà de la phase d'amorçage. Ce principe garantit la reproductibilité et fournit une piste d'audit naturelle.

Arborescence du dépôt GitOps

```
opp-lab-gitops/
├── bootstrap/                # amorçage : Argo CD, projet opp-lab, app-of-apps
├── platform/
│   ├── ingress/  cert-manager/  openbao/  external-secrets/  harbor/
```

```
│   └── policies/             # NetworkPolicies, ResourceQuotas, Kyverno
├── apps/
│   ├── opp-iam/              # Keycloak (realm opp-lab en JSON), OPA (bundles)
│   ├── opp-gateway/          # Gravitee : APIs et plans déclarés
│   ├── opp-streaming/        # Strimzi : Kafka, KafkaTopic, KafkaUser
│   ├── opp-observ/           # OTel Collector, OpenSearch, Data Prepper
│   ├── opp-ledger/  opp-switch/  opp-iso20022/  opp-mojaloop/
│   ├── opp-phee/  opp-orchestr/  opp-workflow/  opp-fraud/
├── environments/
│   ├── level1-compose/       # docker-compose par cas d'usage
│   ├── level2-team/          # valeurs Helm du Lab d'équipe
│   └── level3-perf/          # valeurs HA / performance
└── docs/adr/                 # décisions d'architecture (ADR)
```

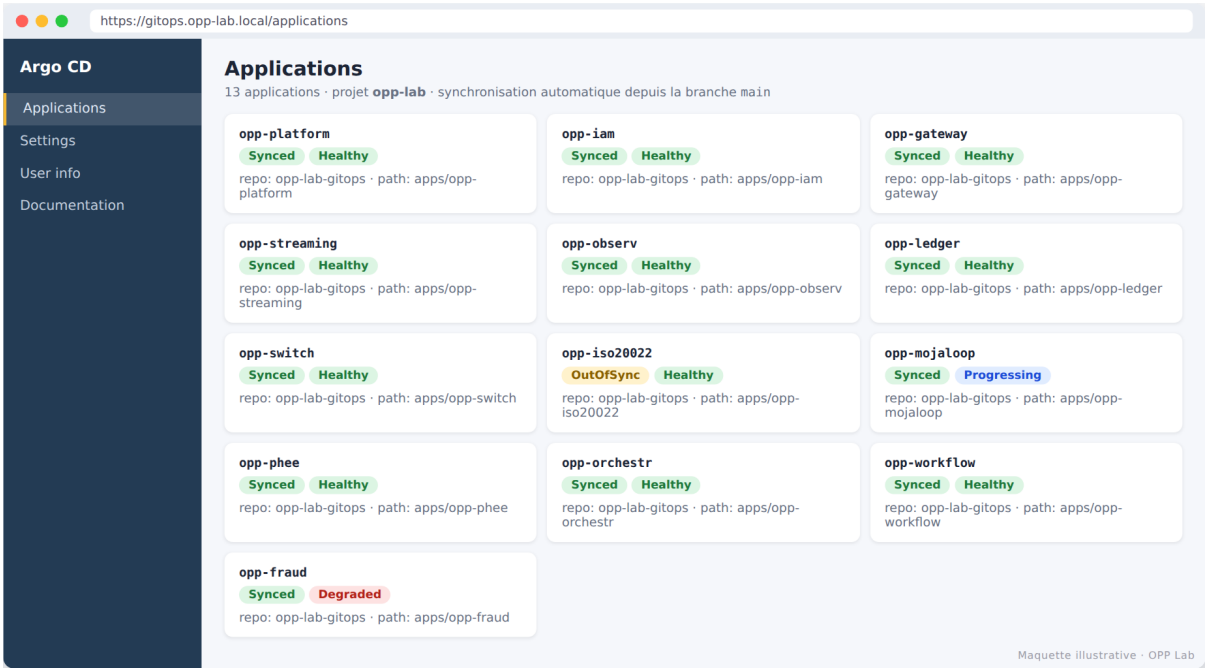


Figure 4 — Argo CD — état de synchronisation des 13 applications du Lab (capture illustrative)

5.5 Gestion des secrets

Aucun secret (mot de passe de base, clé API, certificat, secret client OIDC) n'est stocké dans Git. Les secrets sont conservés dans **OpenBao** (fork open source de HashiCorp Vault, gouverné par la Linux Foundation) et projetés dans Kubernetes par **External Secrets Operator**. Les certificats TLS internes sont émis par cert-manager à partir d'une autorité de certification propre au Lab.

Projection d'un secret OpenBao dans le namespace du ledger

```yaml
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata: { name: ledger-db, namespace: opp-ledger }
spec:
  refreshInterval: 1h
  secretStoreRef: { name: openbao-lab, kind: ClusterSecretStore }
  target: { name: ledger-db-credentials }
  data:
    - secretKey: POSTGRES_PASSWORD
      remoteRef: { key: opp-lab/ledger/postgres, property: password }
```

5.6 Nommage, DNS et points d'accès

Tableau 10 — Points d'accès du Lab — les ports doivent être vérifiés pour chaque version déployée

Service	URL du Lab	Port interne (par défaut)	Accès
Passerelle API (Gravitee)	https://api.opp-lab.local	8082 (gateway)	Public Lab
Console Gravitee	https://apim.opp-lab.local	8083 / 8084	VPN admin
Keycloak	https://iam.opp-lab.local	8080	Public Lab (OIDC) / VPN (admin)
OPA	interne uniquement	8181	Namespaces autorisés
Hyperswitch router / Control Center	https://pay.opp-lab.local	8080 / 9000	Interne / VPN
Temporal (frontend / UI)	https://workflow.opp-lab.local	7233 / 8080	Interne / VPN
Formance Ledger	interne uniquement	8080 (selon image)	Worker de saga seulement
jPOS (ISO 8583)	tcp://switch.opp-lab.local	10000 (configurable)	Simulateurs uniquement
Mojaloop / Testing Toolkit	https://ttk.opp-lab.local	selon charts	VPN
Payment Hub EE Operations	https://ops.phee.opp-lab.local	selon charts	VPN
Tazama TMS API	interne uniquement	5000	Worker de saga, adaptateurs
OpenSearch / Dashboards	https://observ.opp-lab.local	9200 / 5601	VPN (rôle ops)
OTel Collector	interne uniquement	4317 (gRPC) / 4318 (HTTP)	Tous les namespaces

5.7 Données de test synthétiques

Le Lab dispose d'un générateur de données reproductible (graine fixe) qui produit clients, comptes, wallets, commerçants, terminaux et historiques de transactions. Les numéros de téléphone utilisent des plages fictives, les PAN sont des numéros de test valides au sens de Luhn mais non émis, les BIC appartiennent à un espace de noms fictif propre au Lab. Des profils de comportement (clients réguliers, commerçants à forte volumétrie, comptes mules) alimentent les scénarios de fraude.

Règle absolue

Aucune donnée issue d'un système de production — même anonymisée — n'entre dans le Lab sans validation formelle du DPO et du RSSI. L'anonymisation de données de paiement est notoirement difficile ; la génération synthétique est la voie par défaut.

6. Déploiement pas à pas

Le déploiement suit onze phases (0 à 10). Chaque phase possède des prérequis, une procédure, des critères de vérification et une preuve (capture ou sortie de commande) à archiver dans le journal du Lab. L'ordre proposé construit d'abord les services transverses (identité, événements, observabilité) afin que chaque brique ajoutée ensuite soit immédiatement sécurisée et observable.

Tableau 11 — Séquencement du déploiement

Phase	Contenu	Durée indicative	Critère de sortie
0	Socle : Kubernetes, ingress, cert-manager, Argo CD, OpenBao	2 jours	Argo CD synchronise l'app-of-apps
1	Keycloak, OPA, Gravitee	2 jours	Appel API protégé par jeton et politique
2	Kafka, OpenTelemetry, OpenSearch	2 jours	Trace de test visible dans OpenSearch
3	Formance Ledger	1 jour	Transaction Numscript équilibrée
4	jPOS	2 jours	Échange 0100/0110 et 0400/0410 simulé
5	Adaptateur ISO 20022 (Prowide)	2 jours	Aller-retour pacs.008 → pacs.002
6	Mojaloop + Testing Toolkit	3 jours	Collection P2P verte
7	Payment Hub EE	3 jours	Transfert DFSP → Mojaloop orchestré
8	Hyperswitch	2 jours	Paiement routé avec bascule
9	Temporal + workers de saga	2 jours	Saga complète avec compensation
10	Tazama	2 jours	Alerte fraude déclenchée et compensée

Convention

Les commandes ci-dessous sont données à titre de référence. Les noms de charts, de valeurs et les versions évoluent : se référer à la documentation officielle de chaque projet et épingler les versions dans `environments/level2-team/versions.yaml`.

6.1 Phase 0 — Socle

Prérequis : trois machines Linux (Ubuntu Server LTS ou équivalent), réseau privé, résolution DNS interne pour `*.opp-lab.local`, accès à un registre d'images (miroir local recommandé en contexte de connectivité limitée).

Phase 0 — amorçage du socle

```
# Nœud 1 : installation de k3s (serveur) sans l'ingress par défaut
curl -sfL https://get.k3s.io | sh -s - server --cluster-init --disable traefik
# Nœuds 2 et 3 : jonction au cluster
curl -sfL https://get.k3s.io | K3S_URL=https://opp-lab-01:6443 K3S_TOKEN=<token> sh -s - server

# Ingress NGINX, cert-manager, Argo CD
helm upgrade --install ingress-nginx ingress-nginx/ingress-nginx -n ingress --create-namespace
helm upgrade --install cert-manager jetstack/cert-manager -n cert-manager --create-namespace --set
crds.enabled=true
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/
install.yaml
```

```
# Amorçage GitOps : l'application racine pointe sur opp-lab-gitops/apps
kubectl apply -f bootstrap/app-of-apps.yaml
```

Vérification : toutes les applications apparaissent dans Argo CD ; `kubectl get nodes` affiche trois nœuds `Ready` ; un certificat de test est émis par l'autorité interne.

6.2 Phase 1 — Identité, politiques et passerelle

Keycloak est déployé avec un realm `opp-lab` importé depuis Git (clients, rôles, scopes). OPA est déployé en service central et charge ses politiques sous forme de bundle construit par la CI. Gravitee est déployé avec sa base de configuration ; les APIs sont déclarées en code (définitions importées par l'API de gestion).

Phase 1 — IAM, politiques, passerelle

```
# Realm versionné dans Git, importé au démarrage
kubectl -n opp-iam create configmap opp-lab-realm --from-file=apps/opp-iam/realm/opp-lab.json
helm upgrade --install keycloak <chart-keycloak> -n opp-iam -f apps/opp-iam/keycloak-values.yaml

# OPA : serveur de décision + bundle de politiques
helm upgrade --install opa <chart-opa> -n opp-iam -f apps/opp-iam/opa-values.yaml
opa test apps/opp-iam/policies/ -v          # tests unitaires des politiques (CI)

# Gravitee APIM
helm repo add graviteeio https://helm.gravitee.io
helm upgrade --install apim graviteeio/apim -n opp-gateway -f apps/opp-gateway/apim-values.yaml

# Test : obtention d'un jeton puis appel protégé
TOKEN=$(curl -s -d grant_type=client_credentials -d client_id=psp-partner-01 \
  -d client_secret=$SECRET https://iam.opp-lab.local/realms/opp-lab/protocol/openid-connect/token |
jq -r .access_token)
curl -H "Authorization: Bearer $TOKEN" https://api.opp-lab.local/payments/health
```

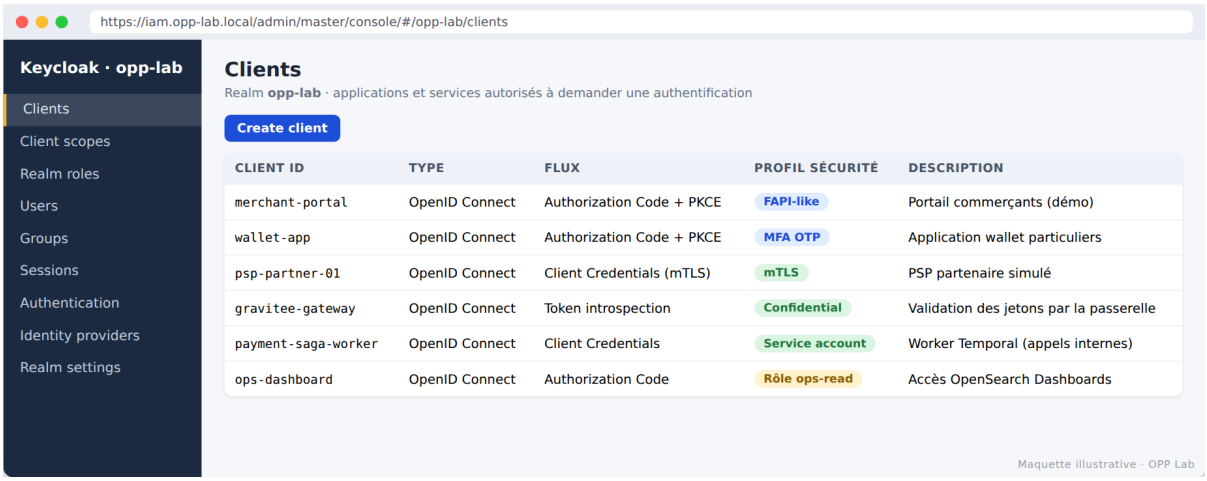


Figure 5 — Keycloak — clients déclarés dans le realm opp-lab (capture illustrative)

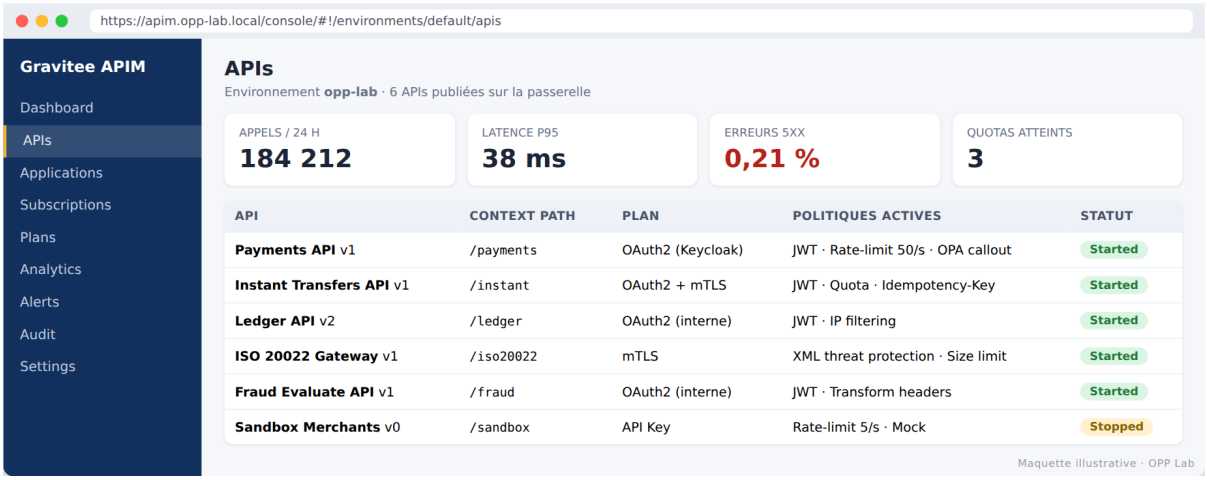


Figure 6 — Gravitee — APIs publiées et politiques actives (capture illustrative)

6.3 Phase 2 — Événements et observabilité

Kafka est déployé avec l'opérateur Strimzi en mode KRaft ; topics et utilisateurs sont déclarés sous forme de ressources Kubernetes (`KafkaTopic`, `KafkaUser`), ce qui les place sous contrôle GitOps. La chaîne d'observabilité comprend un OTel Collector (réception OTLP, enrichissement Kubernetes, masquage), Data Prepper pour la mise en forme des traces, et un cluster OpenSearch avec ses Dashboards.

Déclaration d'un topic sous contrôle GitOps

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: opp.payments.settled.v1
  namespace: opp-streaming
  labels: { strimzi.io/cluster: opp-kafka }
spec:
  partitions: 12
  replicas: 3
  config:
    retention.ms: 1209600000        # 14 jours dans le Lab
    min.insync.replicas: 2
    cleanup.policy: delete
```

OTel Collector — masquage des attributs sensibles avant export

```
receivers:
  otlp: { protocols: { grpc: {}, http: {} } }
processors:
  k8sattributes: {}
  batch: {}
  attributes/mask:
    actions:
      - { key: card.pan, action: delete }
      - { key: party.msisdn, action: hash }
exporters:
  otlp/data-prepper: { endpoint: data-prepper.opp-observ:21890, tls: { insecure: true } }
service:
  pipelines:
    traces: { receivers: [otlp], processors: [k8sattributes, attributes/mask, batch], exporters:
[otlp/data-prepper] }
```

Kafka — topics et événements de paiement

```
ops@opp-lab:~$ kubectl -n opp-streaming exec opp-kafka-broker-0 -- bin/kafka-topics.sh --bootstrap-server localhost:9092 --list
opp.payments.initiated.v1
opp.payments.authorized.v1
opp.payments.settled.v1
opp.payments.failed.v1
opp.ledger.postings.v1
opp.fraud.alerts.v1
opp.iso8583.messages.v1
opp.iso20022.messages.v1
opp.audit.trail.v1
ops@opp-lab:~$ kcat -b opp-kafka-bootstrap:9092 -t opp.payments.settled.v1 -C -o -1 -e | jq
{
  "specversion": "1.0",
  "type": "opp.payment.settled",
  "source": "/opp/workflow/payment-saga",
  "id": "evt-01J9Z6Q4X3K8M2N5P7R1S0T9V",
  "time": "2026-09-08T10:42:17.381Z",
  "traceparent": "00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01",
  "data": {
    "paymentId": "PAY-2026-000418", "rail": "MOJALOOP",
    "amount": { "value": 25000, "currency": "XOF" },
    "debtor": "wallet:+22507000001", "creditor": "wallet:+22507000002",
    "ledgerTxId": 1842, "fraudScore": 0.12, "status": "SETTLED"
  }
}
```

Maquette illustrative · OPP Lab

Figure 7 — Kafka — topics du Lab et événement CloudEvents opp.payment.settled (capture illustrative)

6.4 Phase 3 — Ledger Formance

Le ledger est déployé avec sa base PostgreSQL. Un ledger logique opp est créé, puis le plan de comptes est initialisé par un script d'amorçage (comptes de plateforme, comptes de frais, comptes de règlement par rail, wallets de test).

Phase 3 — initialisation du ledger

```
# Création du ledger logique puis d'une transaction de dotation initiale
curl -X POST $LEDGER/v2/opp
curl -X POST $LEDGER/v2/opp/transactions -H 'Content-Type: application/json' -d '{
  "script": { "plain": "send [XOF 1000000] ( source = @world destination =
@wallets:22507000001 )" },
  "metadata": { "purpose": "lab-seed" }
}'
# Contrôle du solde
curl -s "$LEDGER/v2/opp/accounts/wallets:22507000001?expand=volumes" | jq '.data.volumes'
```

Formance Ledger — écriture atomique Numscript

```
dev@opp-lab:~/ledger$ cat p2p.num
vars {
  monetary $amount
  account  $debtor
  account  $creditor
}
send $amount (
  source = $debtor
  destination = {
    99% to $creditor
    remaining to @platform:fees:p2p
  }
)
dev@opp-lab:~/ledger$ curl -s -X POST $LEDGER/v2/opp/transactions -H "Idempotency-Key: PAY-2026-000418" \
   -d @p2p-request.json | jq '.data | {id, postings}'
{
  "id": 1842,
  "postings": [
    { "source": "wallets:22507000001", "destination": "wallets:22507000002", "amount": 24750, "asset": "XOF" },
    { "source": "wallets:22507000001", "destination": "platform:fees:p2p",    "amount": 250,   "asset": "XOF" }
  ]
}
dev@opp-lab:~/ledger$ curl -s $LEDGER/v2/opp/accounts/wallets:22507000002?expand=volumes | jq '.data.volumes.XOF'
{ "input": 124750, "output": 0, "balance": 124750 }
```

Maquette illustrative · OPP Lab

Figure 8 — Formance — écriture Numscript avec ventilation des frais (capture illustrative)

6.5 Phase 4 — Switch jPOS

Le switch est construit à partir du modèle de projet jPOS (Gradle). La configuration est portée par des descripteurs déposés dans le répertoire `deploy/` du conteneur Q2 : un serveur ISO 8583, un gestionnaire de transactions et ses participants. L'image est construite par la CI, scannée, signée puis déployée par Argo CD.

jPOS — serveur ISO 8583 du Lab

```
<!-- deploy/10_opp_server.xml -->
<server class="org.jpos.q2.iso.QServer" logger="Q2" name="opp-switch">
  <attr name="port" type="java.lang.Integer">10000</attr>
  <channel class="org.jpos.iso.channel.NACChannel" logger="Q2"
           packager="org.jpos.iso.packager.GenericPackager">
    <property name="packager-config" value="cfg/opp-iso87.xml"/>
  </channel>
  <request-listener class="org.jpos.iso.IncomingListener" logger="Q2">
    <property name="queue" value="OPP.TXN"/>
    <property name="ctx.DESTINATION" value="OPP"/>
  </request-listener>
</server>
```

jPOS — chaîne de participants du TransactionManager

```
<!-- deploy/20_opp_txnmgr.xml -->
<txnmgr class="org.jpos.transaction.TransactionManager" logger="Q2">
  <property name="queue" value="OPP.TXN"/>
  <property name="sessions" value="16"/>
  <participant class="lab.opp.switch.PrepareContext"/>
  <participant class="lab.opp.switch.CheckFields"/>
  <participant class="lab.opp.switch.OpaPolicyCheck"/>
  <participant class="lab.opp.switch.FraudScore"/>
  <participant class="lab.opp.switch.LedgerHold"/>
  <participant class="lab.opp.switch.SendResponse"/>
</txnmgr>
```

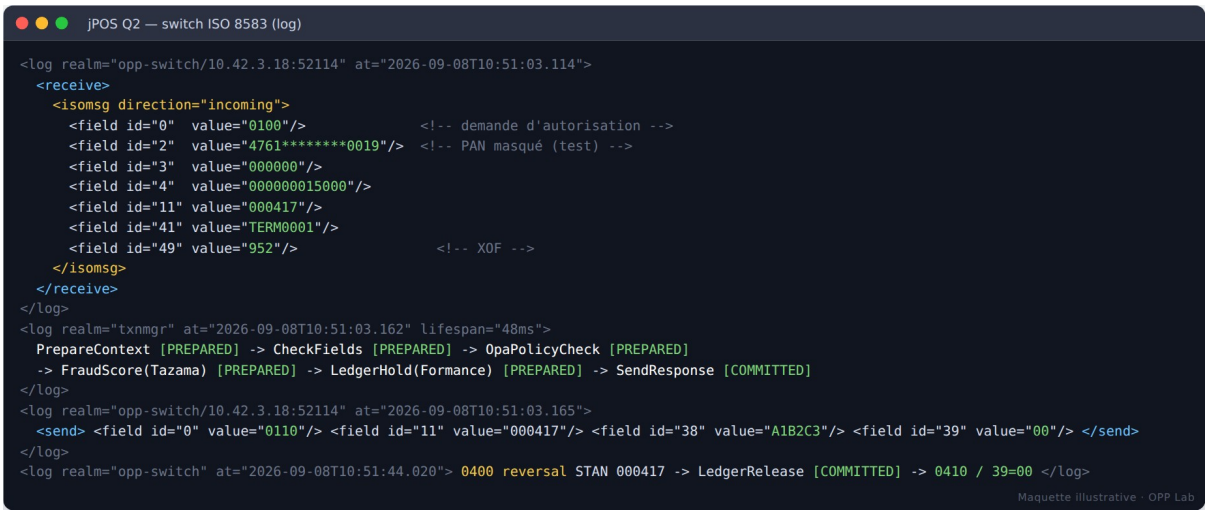


Figure 9 — jPOS — autorisation 0100/0110 puis annulation 0400/0410 (capture illustrative)

6.6 Phase 5 — Adaptateur ISO 20022

L'adaptateur est un microservice Spring Boot qui embarque la bibliothèque Prowide ISO 20022. Il expose des endpoints de réception de messages (pacs.008, pain.001), valide les messages par rapport aux schémas XSD, les convertit vers le modèle canonique et publie les événements correspondants. Sa construction et ses tests sont détaillés au chapitre 7.

pom.xml — dépendance Prowide ISO 20022

```
<dependency>
  <groupId>com.prowidesoftware</groupId>
  <artifactId>pw-iso20022</artifactId>
  <version>${prowide.iso20022.version}</version>  <!-- version SRU épinglée -->
</dependency>
```

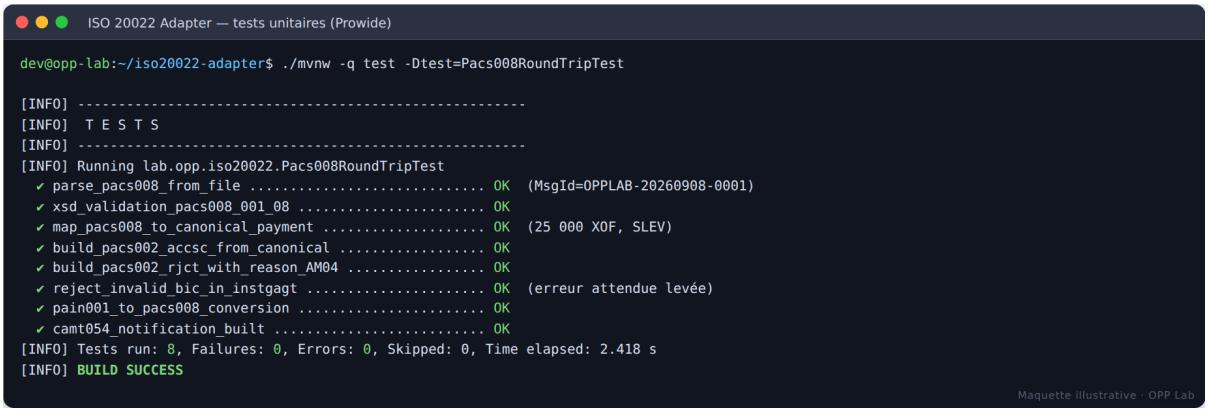


Figure 10 — Adaptateur ISO 20022 — tests d'aller-retour pacs.008 / pacs.002 (capture illustrative)

6.7 Phase 6 — Mojaloop et Testing Toolkit

Mojaloop est déployé par ses charts Helm officiels, précédés de ses dépendances (MySQL, MongoDB, Kafka, Redis) — le Lab peut soit utiliser le chart de dépendances d'exemple fourni par le projet, soit réutiliser son propre cluster Kafka. Pour le niveau 1, des scripts communautaires de déploiement simplifié (tels que mini-loop) permettent d'obtenir une instance sur une seule machine. Le **Mojaloop Testing Toolkit** est ensuite utilisé pour simuler les DFSP et exécuter les collections de tests.

Phase 6 — déploiement Mojaloop (noms de charts à vérifier selon la version)

```
helm repo add mojaloop https://mojaloop.io/helm/repo/
helm repo update
# Dépendances (exemple fourni par le projet) puis Mojaloop
helm upgrade --install backend mojaloop/example-mojaloop-backend -n opp-mojaloop --create-namespace
helm upgrade --install moja mojaloop/mojaloop -n opp-mojaloop -f apps/opp-mojaloop/values-lab.yaml

# Vérification de santé
curl -s http://ml-api-adapter.opp-mojaloop/health | jq .status
# Exécution de la collection P2P du Lab dans le Testing Toolkit (interface web ou CLI)
```

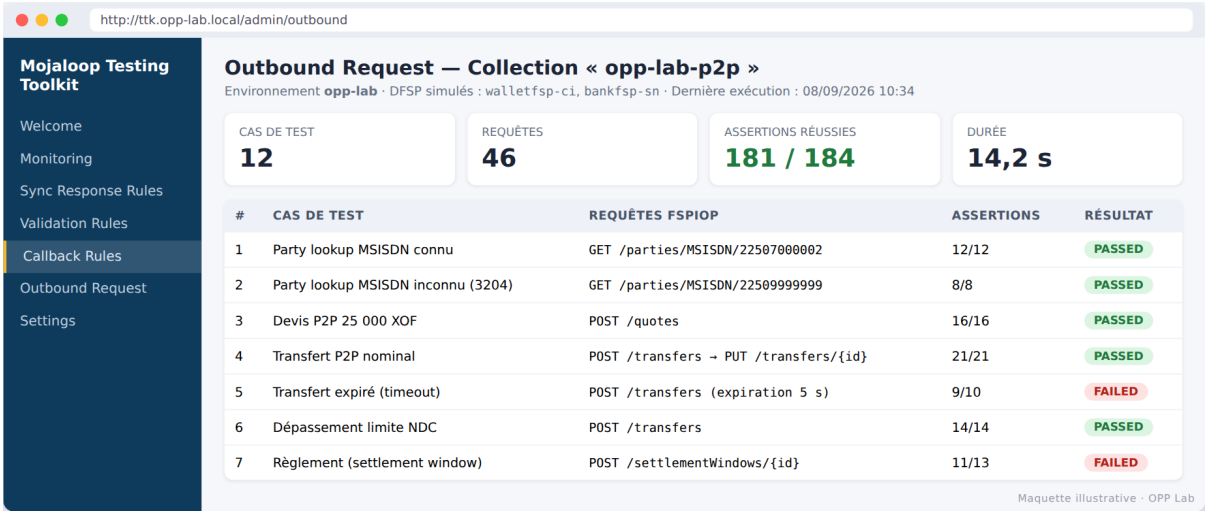


Figure 11 — Mojaloop Testing Toolkit — collection P2P du Lab, 2 cas en échec à analyser (capture illustrative)

Les deux échecs visibles sur la capture sont volontaires : ils correspondent à des scénarios négatifs (expiration et fenêtre de règlement) dont l'analyse est un exercice pédagogique à part entière.

6.8 Phase 7 — Mifos Payment Hub EE

Payment Hub EE est déployé par ses charts Helm (moteur Zeebe, connecteurs, application Operations). Dans le Lab, deux tenants représentent deux DFSP : `walletfsp-ci` (émetteur de monnaie électronique simulé) et `bankfsp-sn` (banque simulée). Chaque tenant est configuré avec son connecteur de gestion de comptes et son identifiant de participant Mojaloop.

Phase 7 — Payment Hub EE

```
 helm upgrade --install ph-ee <chart-ph-ee-engine> -n opp-phee -f apps/opp-phee/values-lab.yaml
 # Déploiement des workflows BPMN (Zeebe) versionnés dans Git
 zbctl deploy apps/opp-phee/bpmn/PayerFundTransfer-DFSPID.bpmn --address ph-ee-zeebe-gateway:26500
--insecure
 # Initiation d'un transfert par l'API canal
 curl -X POST https://channel.phee.opp-lab.local/channel/transfer -H 'Platform-TenantId: walletfsp-
ci' \
   -H 'Content-Type: application/json' -d @samples/phee-p2p-25000.json
```

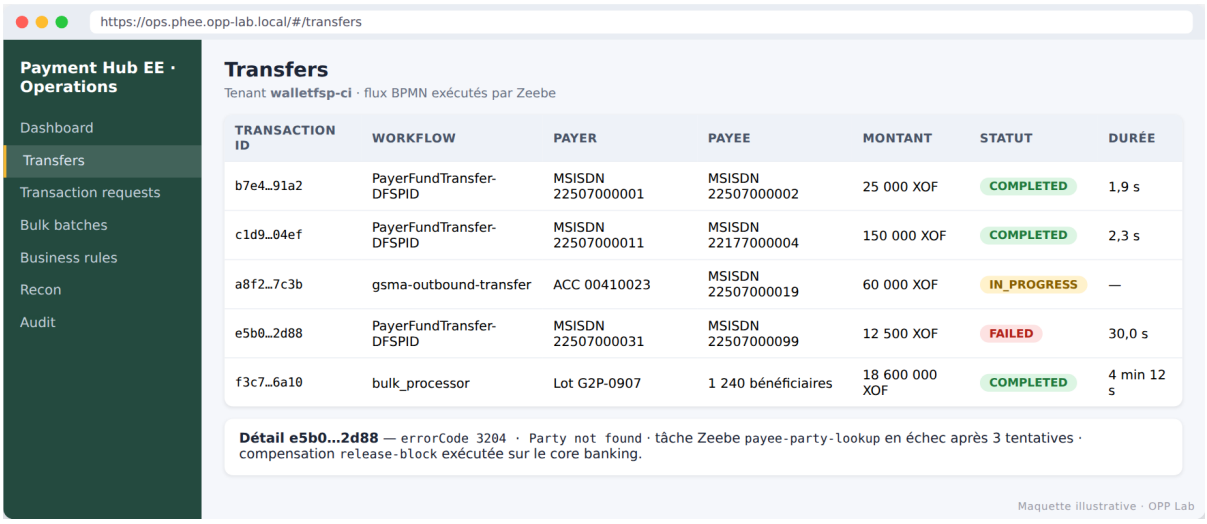


Figure 12 — Payment Hub EE — suivi des transferts orchestrés par Zeebe (capture illustrative)

6.9 Phase 8 — Hyperswitch

Hyperswitch est déployé avec son router, son scheduler, son Control Center, une base PostgreSQL et un cache Redis. Deux connecteurs de test simulent des PSP cartes ; deux connecteurs personnalisés du Lab (`mojaloop-bridge` et `iso20022-adapter`) renvoient vers les rails internes via le workflow Temporal.

Phase 8 — Hyperswitch (se référer au guide officiel pour le déploiement Kubernetes)

```
 # Niveau 1 : dépôt officiel et Docker Compose
 git clone --depth 1 https://github.com/juspay/hyperswitch && cd hyperswitch
 docker compose up -d

 # Création d'un paiement de test (clé API marchand du Lab)
 curl -X POST http://localhost:8080/payments -H "api-key: $HS_API_KEY" -H 'Content-Type:
application/json' -d '{
   "amount": 15000, "currency": "XOF", "confirm": true,
   "payment_method": "card", "customer_id": "cust_lab_0001",
   "metadata": { "paymentId": "PAY-2026-000512" }
 }'
```



Figure 13 — Hyperswitch Control Center — règle de routage du Lab et connecteurs (capture illustrative)

6.10 Phase 9 — Temporal et workers de saga

Temporal fournit l'exécution durable des workflows de paiement : chaque étape est journalisée, les échecs transitoires sont rejoués automatiquement selon une politique de retry, et les compensations sont garanties même en cas de redémarrage des workers. Le Lab déploie le serveur Temporal (chart Helm officiel) et le worker `payment-saga-worker` développé au chapitre 7.

Phase 9 — Temporal

```
# Niveau 1 : serveur de développement tout-en-un
temporal server start-dev --namespace opp-lab --ui-port 8233

# Niveau 2 : chart Helm officiel
helm repo add temporal https://go.temporal.io/helm-charts
helm upgrade --install temporal temporal/temporal -n opp-workflow -f apps/opp-workflow/temporal-values.yaml
temporal operator namespace create opp-lab --address temporal-frontend.opp-workflow:7233
```

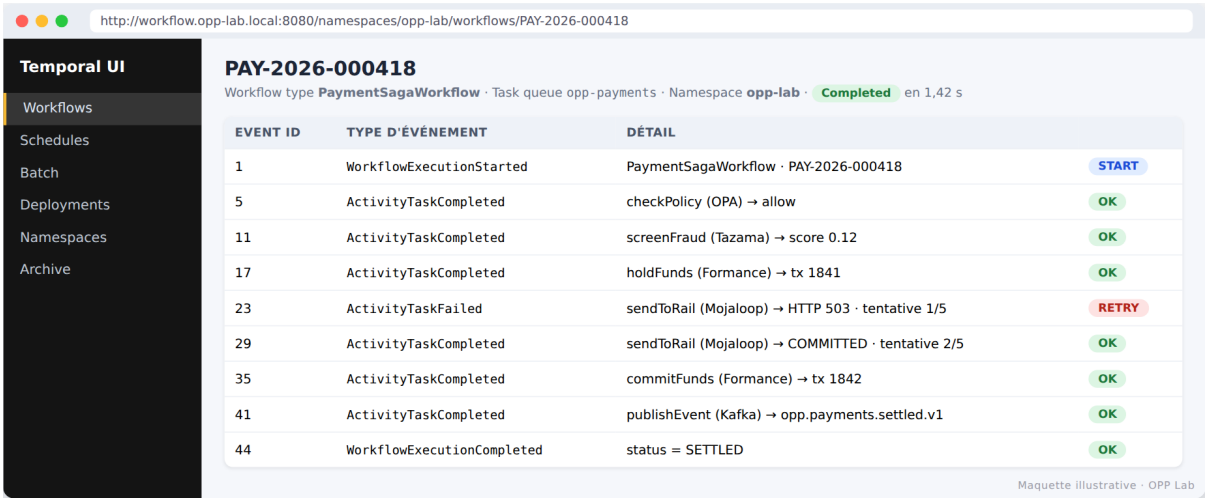


Figure 14 — Temporal UI — historique d'un PaymentSagaWorkflow avec retry automatique (capture illustrative)

6.11 Phase 10 — Tazama

Tazama est déployé à partir de sa pile de référence (services cœur : PostgreSQL, NATS, Valkey ; TMS API ; event director ; processeurs de règles ; processeur de typologies ; TADProc ; services de relais). La configuration — carte réseau (network map), règles et typologies — est chargée au démarrage. Un relais NATS vers Kafka publie les alertes sur `opp.fraud.alerts.v1`.

Phase 10 — Tazama (pile de référence Docker)

```
git clone https://github.com/tazama-lf/Full-Stack-Docker-Tazama && cd Full-Stack-Docker-Tazama
# Suivre le README : choix du profil de déploiement, jeton d'accès aux paquets, lancement
# Test d'évaluation d'une transaction ISO 20022 (chemin selon la version)
curl -X POST http://localhost:5000/v1/evaluate/iso20022/pacs.008.001.10 \
  -H 'Content-Type: application/json' -d @samples/pacs008-nominal.json
```



Figure 15 — Tazama — typologie en alerte (seuil d'interdiction dépassé) (capture illustrative)

6.12 Vérification globale

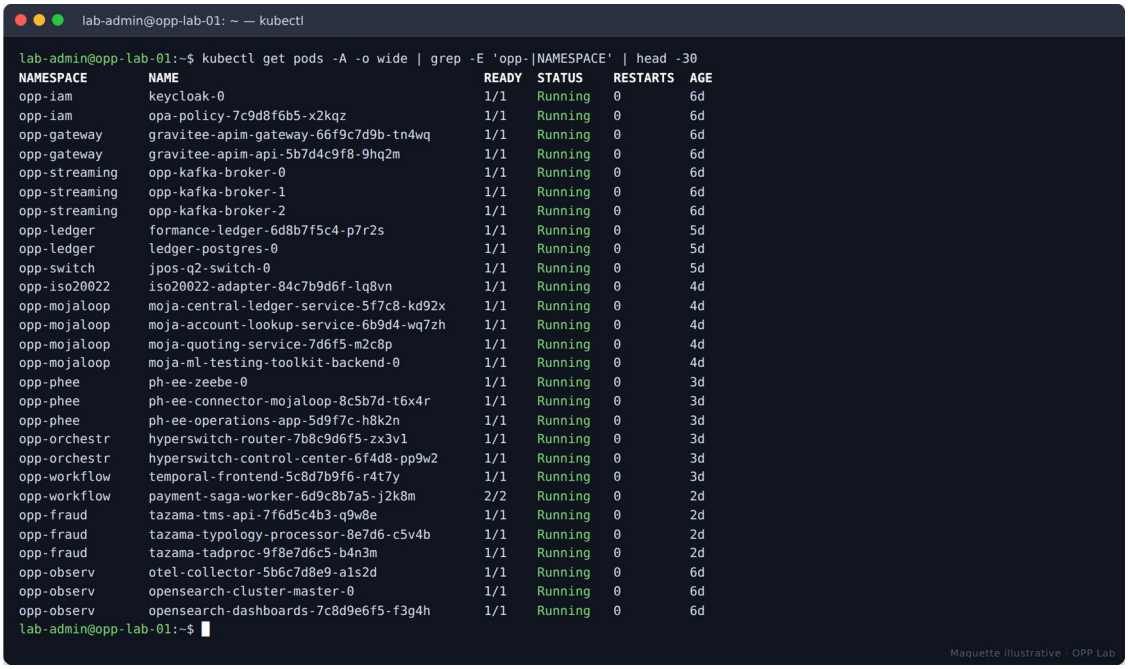


Figure 16 — État des pods du Lab après les onze phases (capture illustrative)

- Toutes les applications Argo CD sont `Synced` et `Healthy` ; les éventuels écarts sont documentés.
- Le scénario de fumée `smoke-e2e` (paiement wallet → wallet de 1 000 XOF) aboutit en moins de 2 secondes.
- La trace correspondante est visible dans OpenSearch avec au moins sept services.
- Le ledger est équilibré : la somme des soldes de tous les comptes, `@world` inclus, est nulle.
- Les tests de politiques OPA, les tests de contrat et les collections TTK sont verts dans la CI.

7. Développement et intégration

Les briques open source fournissent les moteurs ; la valeur du Lab réside dans le **code d'intégration** qui les fait travailler ensemble. Ce chapitre décrit ce code : organisation du dépôt, modèle canonique, contrats d'événements, saga, ledger, adaptateurs ISO, connecteurs, politiques et instrumentation. Les extraits sont volontairement concis et pédagogiques ; ils constituent le point de départ des exercices des équipes.

7.1 Organisation du code applicatif

Arborescence du monorepo applicatif

```
opp-lab-services/                    # monorepo des composants développés pour le Lab
├── contracts/
│   ├── openapi/payments-api-v1.yaml # contrat de la Payments API
│   ├── asyncapi/opp-events-v1.yaml  # contrat des événements Kafka
│   └── schemas/canonical-payment.schema.json
├── canonical-model/                 # bibliothèque Java partagée (records, mappers)
├── payment-saga-worker/             # worker Temporal (workflows + activités)
├── iso20022-adapter/                # Spring Boot + Prowide ISO 20022
├── switch-participants/             # participants jPOS (JAR déposé dans l'image Q2)
├── phee-workers/                    # workers Zeebe complémentaires (fraude, ledger)
├── hyperswitch-connectors/          # connecteurs Lab (mojaloop-bridge, iso20022)
├── policies/                        # politiques Rego + tests
├── simulators/                      # émetteur, acquéreur, PSP, régulateur
├── data-generator/                  # génération de données synthétiques
└── perf/                            # scénarios k6 et JMeter
```

Chaque composant est livré par la même chaîne CI : compilation, tests unitaires, tests de contrat, analyse statique, génération de SBOM (CycloneDX), scan de vulnérabilités, construction d'image, signature, publication dans le registre du Lab, puis mise à jour de la version dans le dépôt GitOps.

7.2 Modèle canonique de paiement

Toutes les briques parlent des langages différents : ISO 8583 pour jPOS, ISO 20022 pour Prowide et Tazama, FSPIOP pour Mojaloop, l'API propre d'Hyperswitch, Numscript pour Formance. Un **modèle canonique** interne évite la multiplication des conversions point à point : chaque adaptateur convertit une seule fois, depuis et vers ce modèle.

canonical-model — structure du paiement canonique

```
public record CanonicalPayment(
    String paymentId,            // identifiant métier unique (ex. PAY-2026-000418)
    String idempotencyKey,       // clé fournie par le canal, propagée jusqu'au ledger
    Rail rail,                   // CARD_ISO8583 | INSTANT_MOJALOOP | ISO20022 | PSP
    Money amount,                // montant en unités mineures + devise ISO 4217
    Party debtor,                // alias, compte ou PAN tokenisé
    Party creditor,
    String purpose,              // motif (catégorie ISO 20022 si disponible)
    Instant createdAt,
    Map<String, String> metadata // endToEndId, STAN, transactionId Mojaloop...
) {}

public record Money(long minorUnits, String currency) {
    public static Money xof(long value) { return new Money(value, "XOF"); } // XOF : 0 décimale
}
```

Tableau 12 — Table de correspondance simplifiée vers le modèle canonique

Champ canonique	ISO 8583	ISO 20022 (pacs.008)	FSPIOP (Mojaloop)	Hyperswitch
paymentId	dérivé STAN + RRN (champs 11, 37)	EndToEndId / TxId	transactionId	metadata.paymentId
amount.minorUnits	champ 4	IntrBkSttlmAmt	amount.amount	amount

Champ canonique	ISO 8583	ISO 20022 (pacs.008)	FSPIOP (Mojaloop)	Hyperswitch
amount.currency	champ 49 (numérique, 952 = XOF)	IntrBkSttlmAmt@Ccy	amount.currency	currency
debtor	champ 2 (PAN tokenisé)	Dbtr / DbtrAcct	payer.partyIdInfo	customer_id / payment_method
creditor	champ 42 (commerçant)	Cdtr / CdtrAcct	payee.partyIdInfo	merchant / profil
statut	champ 39 (code réponse)	pacs.002 TxSts (ACSC, RJCT…)	transferState (COMMITTED…)	status (succeeded…)

7.3 Contrat d'événements Kafka

Les événements respectent la spécification **CloudEvents 1.0** (enveloppe standard : `id`, `source`, `type`, `time`, `specversion`) complétée par l'extension de traçage `traceparent`. La clé de message Kafka est le `paymentId`, ce qui garantit l'ordre des événements d'un même paiement au sein d'une partition.

Tableau 13 — Topics Kafka du Lab

Topic	Producteur	Consommateurs	Rétention Lab
opp.payments.initiated.v1	Worker de saga	Analytique, fraude, audit	14 jours
opp.payments.authorized.v1	Worker de saga, jPOS	Notifications, analytique	14 jours
opp.payments.settled.v1	Worker de saga	Réconciliation, analytique, notifications	14 jours
opp.payments.failed.v1	Worker de saga	Support, analytique, alerting	14 jours
opp.ledger.postings.v1	Worker de saga (post-commit)	Réconciliation, reporting	30 jours
opp.fraud.alerts.v1	Relais Tazama	Worker de saga, case management	30 jours
opp.iso8583.messages.v1	jPOS (messages masqués)	Audit, rejeu de tests	7 jours
opp.iso20022.messages.v1	Adaptateur ISO 20022	Audit, rejeu de tests	7 jours
opp.audit.trail.v1	Tous	Archivage OpenSearch	90 jours

Publication d'un événement CloudEvents (SDK CloudEvents Java)

```
@Component
public class PaymentEventPublisher {
  private final KafkaTemplate<String, CloudEvent> kafka;
  public void settled(CanonicalPayment p, long ledgerTxId) {
    CloudEvent evt = CloudEventBuilder.v1()
        .withId(Ulid.next())
        .withSource(URI.create("/opp/workflow/payment-saga"))
        .withType("opp.payment.settled")
        .withTime(OffsetDateTime.now(ZoneOffset.UTC))
        .withExtension("traceparent", Tracing.currentTraceparent())
        .withData("application/json", Json.bytes(SettledData.of(p, ledgerTxId)))
        .build();
    kafka.send("opp.payments.settled.v1", p.paymentId(), evt); // clé = paymentId
  }
}
```

7.4 Saga de paiement avec Temporal

Le cœur du Lab est le workflow `PaymentSagaWorkflow`. Il enchaîne les étapes du paiement comme un programme séquentiel, tandis que Temporal assure la durabilité : si le worker s'arrête au milieu d'un paiement, un autre worker reprend exactement là où l'exécution s'était arrêtée. Les compensations sont enregistrées au fil de l'eau et exécutées dans l'ordre inverse en cas d'échec.

payment-saga-worker — interfaces du workflow et des activités

```
@WorkflowInterface
public interface PaymentSagaWorkflow {
  @WorkflowMethod PaymentResult execute(CanonicalPayment payment);
  @QueryMethod    String status();
}

@ActivityInterface
public interface PaymentActivities {
  PolicyDecision checkPolicy(CanonicalPayment p);   // OPA
  FraudDecision  screenFraud(CanonicalPayment p);   // Tazama
  String         holdFunds(CanonicalPayment p);     // Formance : débiteur -> holds
  void           releaseFunds(String holdId);       // compensation
  RailResult     sendToRail(CanonicalPayment p);    // Mojaloop | jPOS | ISO 20022 | PSP
  long           commitFunds(String holdId, CanonicalPayment p);
  void           publishEvent(String type, CanonicalPayment p, Object data);
}
```

Implémentation de la saga — la compensation est garantie par Temporal

```
public class PaymentSagaWorkflowImpl implements PaymentSagaWorkflow {
  private String status = "RECEIVED";
  private final PaymentActivities act = Workflow.newActivityStub(PaymentActivities.class,
      ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(10))
          .setRetryOptions(RetryOptions.newBuilder()
              .setInitialInterval(Duration.ofMillis(500))
              .setBackoffCoefficient(2.0)
              .setMaximumAttempts(5)
              .setDoNotRetry("PolicyDenied", "FraudBlocked", "InsufficientFunds")
              .build())
          .build());

  @Override public PaymentResult execute(CanonicalPayment p) {
    Saga saga = new Saga(new Saga.Options.Builder().setParallelCompensation(false).build());
    try {
      if (!act.checkPolicy(p).allow()) return fail(p, "POLICY_DENIED");
      if (act.screenFraud(p).blocked())  return fail(p, "FRAUD_BLOCKED");
      String holdId = act.holdFunds(p);
      saga.addCompensation(act::releaseFunds, holdId);   // enregistrée AVANT l'étape risquée
      status = "FUNDS_HELD";
      RailResult r = act.sendToRail(p);                  // retries automatiques si 5xx / timeout
      if (!r.accepted()) { saga.compensate(); return fail(p, r.reason()); }
      long txId = act.commitFunds(holdId, p);
      status = "SETTLED";
      act.publishEvent("opp.payment.settled", p, Map.of("ledgerTxId", txId));
      return PaymentResult.settled(p.paymentId(), txId);
    } catch (ActivityFailure e) {
      saga.compensate();                                 // libération des fonds garantie
      return fail(p, "TECHNICAL_FAILURE");
    }
  }
  @Override public String status() { return status; }
  private PaymentResult fail(CanonicalPayment p, String reason) {
    status = "FAILED"; act.publishEvent("opp.payment.failed", p, Map.of("reason", reason));
    return PaymentResult.failed(p.paymentId(), reason);
  }
}
```

Point pédagogique clé

Le workflow Temporal doit rester **déterministe** : pas d'appel réseau, pas d'horloge système, pas d'aléatoire direct dans le code du workflow. Tout effet de bord passe par une activité. L'identifiant du workflow est le `paymentId`, ce qui empêche nativement le démarrage de deux sagas pour le même paiement (idempotence au niveau de l'orchestration).

7.5 Ledger : plan de comptes et Numscript

Le plan de comptes du Lab adopte une convention hiérarchique lisible. Le mécanisme de réservation repose sur un compte de transit par paiement (`holds:<paymentId>`), qui rend visible à tout instant l'argent « en vol ».

Tableau 14 — Plan de comptes du Lab

Compte	Nature	Usage
@world	Technique Formance	Source et puits externes (émission, dotation de test)
wallets:<msisdn>	Passif client	Wallets de monnaie électronique simulés
accounts:<iban-lab>	Passif client	Comptes bancaires simulés
holds:<paymentId>	Transit	Fonds réservés pendant l'exécution de la saga
settlement:mojaloop / :card / :iso20022	Règlement	Positions vis-à-vis des rails
platform:fees:<produit>	Produit	Frais perçus par la plateforme
platform:suspense	Attente	Écarts de réconciliation à analyser

Numscript — réservation, règlement et compensation

```
// hold.num — réservation
vars { monetary $amount  account $debtor  account $hold }
send $amount ( source = $debtor  destination = $hold )

// commit.num — règlement avec frais de 1 % vers la plateforme
vars { monetary $amount  account $hold  account $creditor }
send $amount (
  source = $hold
  destination = { 99% to $creditor  remaining to @platform:fees:p2p }
)

// release.num — compensation : restitution intégrale au débiteur
vars { monetary $amount  account $hold  account $debtor }
send $amount ( source = $hold  destination = $debtor )
```

Chaque appel au ledger porte un en-tête d'idempotence égal à `<paymentId>:<étape>` (par exemple `PAY-2026-000418:hold`). En cas de rejeu d'une activité par Temporal, le ledger renvoie la transaction existante au lieu d'en créer une nouvelle : l'idempotence est ainsi garantie à deux niveaux complémentaires.

7.6 Adaptateur ISO 20022 avec Prowide

iso20022-adapter — réception d'un pacs.008 et réponse pacs.002

```
@PostMapping(value = "/iso20022/pacs.008", consumes = "application/xml")
public ResponseEntity<String> receivePacs008(@RequestBody String xml) {
  xsdValidator.validate(xml, "pacs.008.001.08.xsd");       // javax.xml.validation
  MxPacs00800108 mx = MxPacs00800108.parse(xml);
  CreditTransferTransaction39 tx = mx.getFIToFICstmrCdtTrf().getCdtTrfTxInf().get(0);

  CanonicalPayment p = new CanonicalPayment(
      tx.getPmtId().getEndToEndId(),
      tx.getPmtId().getTxId(),
      Rail.ISO20022,
      new Money(tx.getIntrBkSttlmAmt().getValue().longValueExact(),
tx.getIntrBkSttlmAmt().getCcy()),
      PartyMapper.from(tx.getDbtr(), tx.getDbtrAcct()),
      PartyMapper.from(tx.getCdtr(), tx.getCdtrAcct()),
      PurposeMapper.from(tx.getPurp()), Instant.now(), Map.of("msgId",
mx.getFIToFICstmrCdtTrf().getGrpHdr().getMsgId()));

  PaymentResult r = sagaClient.startAndWait(p);             // démarre PaymentSagaWorkflow
  String pacs002 = Pacs002Builder.fromResult(mx, r);        // ACSC ou RJCT + code motif
  return ResponseEntity.ok().contentType(MediaType.APPLICATION_XML).body(pacs002);
```

```
}
```

La méthode `longValueExact()` est volontairement stricte : un montant XOF comportant des décimales lève une exception, ce qui constitue un contrôle métier utile. Le constructeur de pacs.002 s'appuie sur les classes générées de la version de message retenue ; les noms exacts des classes dépendent de la version (par exemple `.001.08` ou `.001.10`) et doivent être alignés sur le guide d'implémentation simulé.

7.7 Participants jPOS

switch-participants — réservation de fonds dans Formance depuis jPOS

```
public class LedgerHold implements TransactionParticipant, AbortParticipant, Configurable {
  private LedgerClient ledger;
  public void setConfiguration(Configuration cfg) { ledger = new LedgerClient(cfg.get("ledger-
url")); }

  public int prepare(long id, Serializable context) {
    Context ctx = (Context) context;
    ISOMsg req = ctx.get("REQUEST");
    try {
      long amount = Long.parseLong(req.getString(4));          // XOF : pas de décimale
      String holdId = ledger.hold(ctx.getString("PAYMENT_ID"), Money.xof(amount),
                                  "cards:" + ctx.getString("TOKEN"));
      ctx.put("HOLD_ID", holdId);
      return PREPARED;
    } catch (InsufficientFunds e) {
      ctx.put("RC", "51");                                     // fonds insuffisants
      return ABORTED | NO_JOIN;
    }
  }
  public void commit(long id, Serializable context) { /* rien : règlement différé (0220 / clearing)
*/ }
  public void abort(long id, Serializable context) {
    Context ctx = (Context) context;
    if (ctx.get("HOLD_ID") != null) ledger.release(ctx.getString("HOLD_ID"));  // compensation
locale
  }
}
```

Le TransactionManager de jPOS offre nativement une sémantique en deux phases (prepare / commit / abort) à l'intérieur du switch. Le Lab met en évidence la différence avec la saga Temporal : ici, la compensation est locale et synchrone ; pour les flux longs impliquant plusieurs systèmes externes, le switch délègue à la saga.

7.8 Workers Zeebe pour Payment Hub EE

phee-workers — insertion d'un contrôle Tazama dans un flux BPMN

```
zeebeClient.newWorker()
    .jobType("opp-fraud-screen")
    .handler((client, job) -> {
        Map<String, Object> v = job.getVariablesAsMap();
        FraudDecision d = tazama.evaluate(Pacs008Mapper.fromPhee(v));
        client.newCompleteCommand(job.getKey())
              .variables(Map.of("fraudBlocked", d.blocked(), "fraudScore", d.score()))
              .send().join();
    })
    .timeout(Duration.ofSeconds(5))
    .open();
```

L'ajout d'une tâche de service `opp-fraud-screen` dans le diagramme BPMN, suivie d'une passerelle exclusive sur la variable `fraudBlocked`, suffit à intégrer la détection de fraude dans les flux de Payment Hub EE sans modifier ses connecteurs existants.

7.9 Connecteurs Hyperswitch du Lab

Hyperswitch intègre ses connecteurs dans son code source (Rust), selon une structure normalisée : transformation de la requête, appel HTTP, transformation de la réponse, gestion des webhooks. Le dépôt fournit un outillage de génération de squelette de connecteur. Le Lab développe deux connecteurs : `mojaloop-bridge`, qui transforme un paiement wallet en démarrage de saga vers Mojaloop, et `iso20022-adapter`, qui fait de même vers l'adaptateur ISO 20022. Ces connecteurs sont d'excellents candidats à une **contribution en amont**, qui renforcerait la présence des rails africains dans l'écosystème.

- **Étape 1** : générer le squelette du connecteur et déclarer ses moyens de paiement supportés.
- **Étape 2** : implémenter les transformations (requête Hyperswitch → API du bridge Lab ; réponse → statut Hyperswitch).
- **Étape 3** : gérer l'asynchronisme — le bridge renvoie `pending`, puis notifie le statut final par webhook.
- **Étape 4** : écrire les tests de connecteur et les jeux de cartes / wallets de test.
- **Étape 5** : déclarer le connecteur dans le Control Center et l'intégrer à une règle de routage.

7.10 Intégration Tazama

Tazama évalue des messages ISO 20022. L'intégration consiste à transformer chaque paiement canonique en pacs.008 (demande) puis pacs.002 (statut), avec des identifiants de parties stables d'une transaction à l'autre — condition indispensable au calcul des règles de vélocité et d'historique. Le résultat (score de typologie, statut `NALT` sans alerte ou `ALRT` en alerte) est relayé sur Kafka et consommé par la saga.

Tableau 15 — Modes d'intégration de la détection de fraude

Situation	Mode d'intégration	Effet sur la latence	Usage recommandé
Paiement instantané de faible montant	Évaluation asynchrone (post-autorisation)	Nulle sur le parcours client	Détection, alerte, investigation
Paiement de montant élevé ou nouveau bénéficiaire	Évaluation synchrone avec seuil d'interdiction	+ quelques dizaines de ms (à mesurer)	Interdiction en ligne
Paiement de masse (G2P, salaires)	Évaluation par lot sur le fichier	Hors parcours	Contrôle préalable au décaissement

7.11 Politiques OPA

policies/payment/limits.rego — plafonds par niveau KYC (valeurs illustratives)

```
package opp.payment
import rego.v1

default allow := false

allow if {
    input.channel in {"wallet", "merchant", "psp"}
    input.amount.value <= data.limits[input.customer.kyc_level].per_tx
    not sanctioned
    not outside_business_hours_for_high_value
}

reasons contains msg if {
    input.amount.value > data.limits[input.customer.kyc_level].per_tx
    msg := sprintf("amount %d %s exceeds per_tx limit %d for kyc_level %s",
        [input.amount.value, input.amount.currency,
         data.limits[input.customer.kyc_level].per_tx, input.customer.kyc_level])
}

sanctioned if input.creditor.country in data.sanctions.countries

outside_business_hours_for_high_value if {
```

```
    input.amount.value > 5000000
    not input.context.business_hours
}
```

Test unitaire de politique — exécuté par « opa test » dans la CI

```
package opp.payment_test
import rego.v1

test_l1_over_limit_denied if {
    not data.opp.payment.allow with input as {"channel": "wallet",
        "amount": {"value": 750000, "currency": "XOF"},
        "customer": {"kyc_level": "L1"}, "creditor": {"country": "CI"}}
        with data.limits as {"L1": {"per_tx": 500000}}
        with data.sanctions.countries as []
}
```



Figure 17 — OPA — refus motivé d'un paiement dépassant le plafond KYC (capture illustrative)

Les plafonds, listes et règles horaires sont des **données** (fichiers JSON du bundle), distinctes de la **logique** (Rego). Les équipes conformité peuvent ainsi proposer une évolution de plafond par merge request, revue et tracée, sans toucher au code des services.

7.12 Instrumentation OpenTelemetry

Activation de l'agent Java OpenTelemetry

```
# Extrait du Deployment du worker de saga (instrumentation automatique Java)
env:
  - { name: JAVA_TOOL_OPTIONS, value: "-javaagent:/otel/opentelemetry-javaagent.jar" }
  - { name: OTEL_SERVICE_NAME, value: "payment-saga-worker" }
  - { name: OTEL_EXPORTER_OTLP_ENDPOINT, value: "http://otel-collector.opp-observ:4317" }
  - { name: OTEL_RESOURCE_ATTRIBUTES, value: "deployment.environment=opp-lab,service.namespace=opp"
}
```

Span métier manuel avec attributs normalisés opp.*

```
Span span = tracer.spanBuilder("ledger.hold").setSpanKind(SpanKind.CLIENT).startSpan();
try (Scope s = span.makeCurrent()) {
  span.setAttribute("opp.payment_id", p.paymentId());
  span.setAttribute("opp.rail", p.rail().name());
  span.setAttribute("opp.amount_minor", p.amount().minorUnits());   // jamais de PAN ni d'alias en
clair
  return ledger.hold(p);
} catch (Exception e) { span.recordException(e); span.setStatus(StatusCode.ERROR); throw e; }
finally { span.end(); }
```

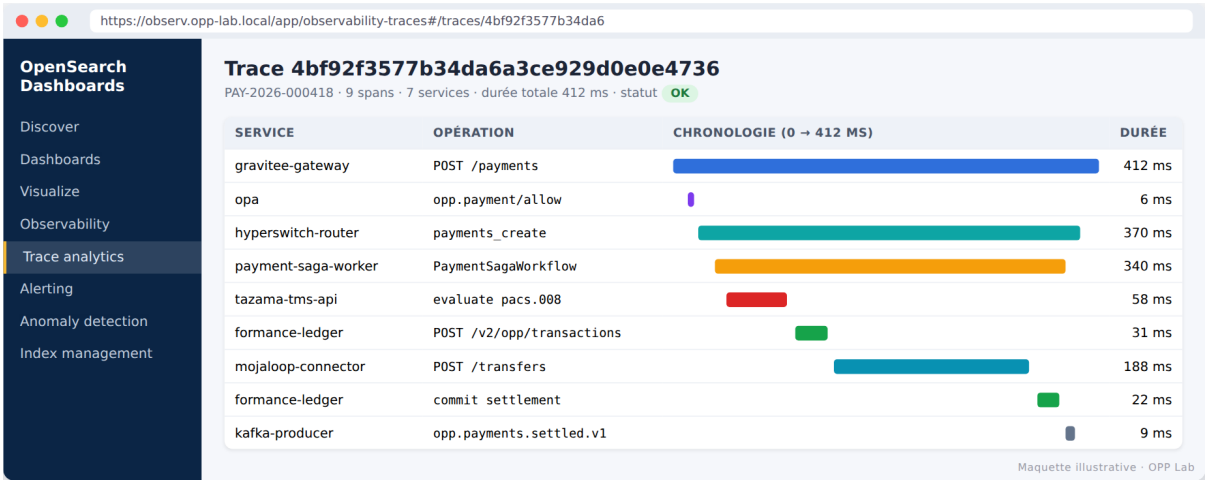


Figure 18 — Trace distribuée d'un paiement à travers sept services (capture illustrative)

7.13 Conventions transverses

Tableau 16 — Conventions transverses de développement

Sujet	Convention du Lab
Idempotence	En-tête `Idempotency-Key` obligatoire sur toute requête mutante ; conservation 24 h ; même clé + corps différent = erreur 422.
Erreurs	Format RFC 9457 (Problem Details) : `type`, `title`, `status`, `detail`, `paymentId`, code métier normalisé.
Délais	Chaque appel sortant a un timeout explicite ; la somme des timeouts d'une saga est inférieure au délai d'expiration du rail.
Versionnage	Versions majeures dans le chemin (`/v1`) et le nom des topics (`.v1`) ; coexistence de deux versions pendant une migration.
Montants	Entiers en unités mineures + devise ISO 4217 ; aucun type flottant dans le code monétaire.
Horodatage	UTC ISO 8601 partout ; conversion en heure locale uniquement à l'affichage.
Identifiants	ULID pour les événements ; `paymentId` lisible et unique ; corrélation par `traceparent`.

8. Cas d'usage testés

Sept cas d'usage de référence structurent la validation du Lab. Chacun est automatisé (collection de tests rejouable), observé (trace et tableau de bord) et documenté (fiche de résultat). Ils couvrent les principaux rails et les principaux mécanismes de contrôle.

8.1 Cas d'usage 1 — Paiement instantané wallet → wallet

Élément	Description
Objectif	Valider le parcours nominal complet d'un transfert P2P instantané entre deux wallets de DFSP différents
Briques	Keycloak, Gravitee, OPA, Hyperswitch, Temporal, Tazama, Formance, Mojaloop, Kafka, OTel / OpenSearch
Préconditions	Wallets dotés dans Formance ; alias enregistrés dans l'account lookup Mojaloop ; règle de routage wallet → mojaloop-bridge
Résultat attendu	Statut SETTLED en moins de 2 s ; écritures hold puis commit équilibrées ; événement settled publié ; trace complète

Déroulé

1. Obtenir un jeton pour `wallet-app` et appeler `POST /payments` (25 000 XOF) avec une clé d'idempotence.
2. Vérifier dans Temporal l'exécution du workflow `PAY-...` et le passage par chaque activité.
3. Contrôler dans Mojaloop (TTK ou journaux) les phases lookup, quote et transfer.
4. Vérifier dans Formance les deux transactions (hold, commit) et les soldes des deux wallets.
5. Rejouer exactement la même requête : aucune nouvelle écriture ne doit être créée (idempotence).

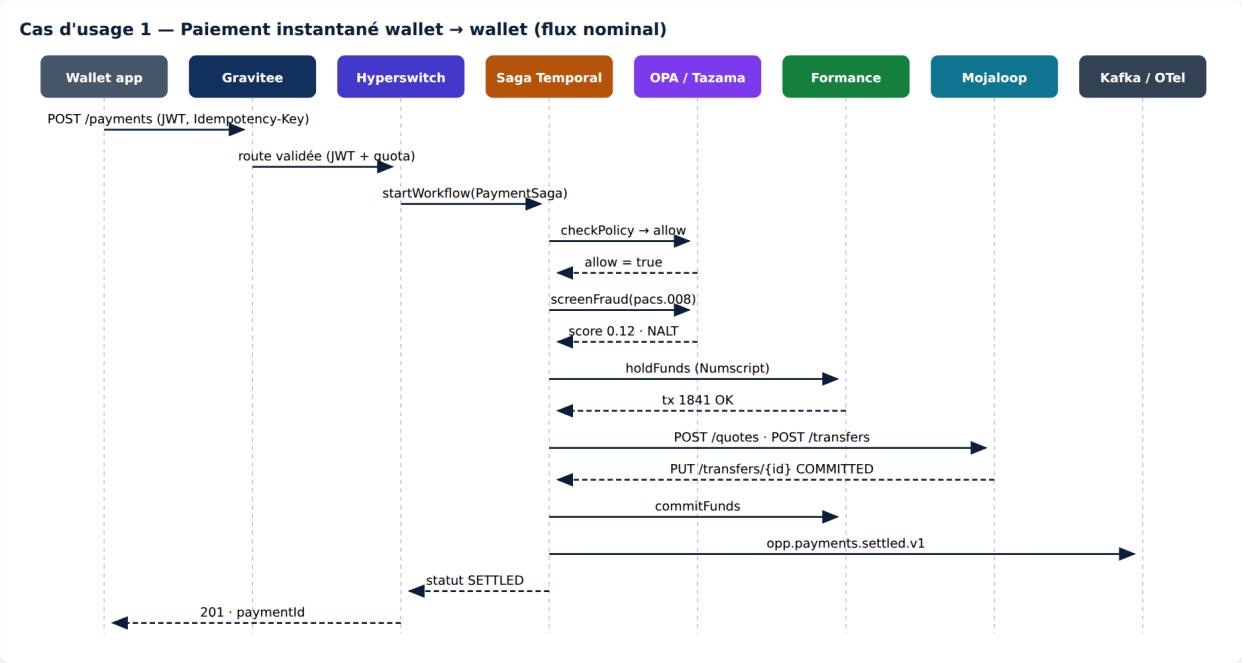


Figure 19 — Diagramme de séquence du cas d'usage 1

8.2 Cas d'usage 2 — Transaction carte ISO 8583 (autorisation → annulation)

Élément	Description
Objectif	Maîtriser le cycle ISO 8583 : autorisation, réponse, puis annulation (reversal) et restitution des fonds
Briques	jPOS, OPA, Tazama, Formance, Kafka, OTel / OpenSearch
Préconditions	Simulateur de terminal / acquéreur ; PAN de test tokenisé ; packager ISO 8583 du Lab
Résultat attendu	0110 avec code 00 et numéro d'autorisation ; puis 0410 code 00 ; hold créé puis libéré ; solde initial restauré

Déroulé

1. Émettre un 0100 de 15 000 XOF depuis le simulateur de terminal.
2. Observer la chaîne de participants dans les journaux Q2 et la trace associée.
3. Émettre un 0400 référençant le même STAN ; vérifier la libération des fonds.
4. Tester les variantes négatives : fonds insuffisants (39 = 51), PAN inconnu, timeout acquéreur suivi d'une annulation automatique.

8.3 Cas d'usage 3 — Connexion d'un PSP via Payment Hub EE

Élément	Description
Objectif	Démontrer l'intégration d'un fournisseur de paiement tiers et d'un core banking par connecteurs et workflows BPMN
Briques	Mifos Payment Hub EE (Zeebe, connecteurs), Mojaloop, Tazama (worker opp-fraud-screen), Kafka
Préconditions	Tenants walletfsp-ci et bankfsp-sn configurés ; workflow BPMN enrichi du contrôle fraude déployé
Résultat attendu	Transfert COMPLETED visible dans Operations ; en cas d'échec de lookup, compensation release-block exécutée

Déroulé

1. Initier un transfert par l'API canal du tenant `walletfsp-ci`.
2. Suivre l'instance BPMN : lookup, contrôle fraude, quote, transfer, notification.
3. Provoquer un échec (MSISDN inconnu) et vérifier l'exécution de la compensation côté core banking.
4. Lancer un lot de paiements de masse (G2P) de 1 000 bénéficiaires et mesurer la durée de traitement.

8.4 Cas d'usage 4 — Simulation PI-SPI (ISO 20022)

Élément	Description
Objectif	Reproduire la logique d'un système de paiement instantané interopérable fondé sur ISO 20022 et la résolution d'alias
Briques	Adaptateur ISO 20022 (Prowide), Temporal, Formance, Tazama, simulateur « opérateur central »
Préconditions	Guide d'implémentation simulé du Lab (profil de messages, codes motifs) ; participants fictifs
Résultat attendu	Échanges pacs.008 / pacs.002 conformes au XSD ; rejets motivés ; notification camt.054 ; transactions irrévocables après validation

Déroulé

1. Émettre un pacs.008 depuis le simulateur de participant émetteur.
2. Vérifier la validation XSD et la conversion vers le modèle canonique.
3. Constater la réponse pacs.002 ACSC et la notification camt.054 au bénéficiaire.
4. Tester les rejets : devise invalide, montant décimal en XOF, BIC inconnu, doublon de MsgId.

Portée de la simulation

Lancée par la BCEAO le 30 septembre 2025, la PI-SPI est la plateforme interopérable de paiement instantané de l'UEMOA ; elle fonctionne en continu, permet l'usage d'alias comme le numéro de téléphone et s'appuie sur ISO 20022. Le Lab n'en reproduit ni les spécifications ni le fonctionnement réel : il en simule les principes afin de préparer les équipes. Toute connexion effective relève exclusivement du processus d'homologation défini par la BCEAO.

8.5 Cas d'usage 5 — Scénarios de fraude avec Tazama

Élément	Description
Objectif	Valider la détection en temps réel, les seuils d'alerte / d'interdiction et la compensation automatique
Briques	Tazama, Temporal, Formance, Kafka, OpenSearch (alerting)
Préconditions	Règles et typologies du Lab chargées ; générateur de comportements frauduleux (rafales, comptes mules)
Résultat attendu	Alerte ALRT sur les scénarios frauduleux ; interdiction au-delà du seuil ; fonds libérés ; alerte visible dans OpenSearch

Déroulé

1. Injecter un historique normal pour un client (30 jours simulés).
2. Lancer une rafale de transferts vers un bénéficiaire nouvellement créé.
3. Vérifier le calcul de typologie et la décision d'interdiction.
4. Contrôler la compensation dans Temporal et l'événement `opp.fraud.alerts.v1`.

8.6 Cas d'usage 6 — Suivi bout en bout dans OpenSearch

Élément	Description
Objectif	Retrouver en moins d'une minute tout l'historique d'un paiement à partir de son identifiant
Briques	OpenTelemetry, Data Prepper, OpenSearch, Kafka (audit trail)
Préconditions	Instrumentation active sur tous les services ; propagation du traceparent dans Kafka
Résultat attendu	Une trace unique regroupe tous les spans ; journaux et événements corrélés par paymentId

Déroulé

1. Rechercher `PAY-2026-000418` dans OpenSearch Dashboards.
2. Ouvrir la trace et identifier le span le plus long.
3. Rebondir vers les journaux du service concerné et vers l'événement Kafka associé.

8.7 Cas d'usage 7 — Tableaux de bord temps réel

Élément	Description
Objectif	Disposer d'une vue opérationnelle et métier en temps réel du Lab
Briques	OpenSearch Dashboards, OTel (métriques), Kafka (événements métier)
Préconditions	Index opp-payments-* alimenté par les événements ; métriques de latence par service
Résultat attendu	Volumes, taux de succès, latence p95, répartition par rail et causes d'échec actualisés en continu

Déroulé

1. Importer le tableau de bord de référence du dépôt (objets sauvegardés).
2. Lancer le générateur de charge pendant 30 minutes.
3. Vérifier la cohérence des volumes affichés avec le ledger (réconciliation).



Figure 20 — Tableau de bord « Suivi bout en bout des paiements » (capture illustrative)

9. Tests, simulation et benchmark

9.1 Pyramide de tests du Lab

Tableau 17 — Pyramide de tests

Niveau	Portée	Outils	Déclenchement
Unitaires	Mappers, participants jPOS, activités, politiques Rego	JUnit, opa test	À chaque commit
Contrat	Respect des contrats OpenAPI / AsyncAPI entre composants	Tests de contrat, validation de schémas	À chaque merge request
Intégration	Deux ou trois briques réelles (ex. saga + ledger)	Testcontainers, Compose de niveau 1	À chaque merge request
Bout en bout	Les sept cas d'usage sur le Lab de niveau 2	Collections TTK, scripts API, simulateurs	Quotidien + avant démonstration
Non fonctionnels	Performance, résilience, sécurité	k6, JMeter, Chaos Mesh, ZAP, Trivy	Hebdomadaire / campagne

9.2 Simulateurs

Un Lab de paiement ne vaut que par la qualité de ses simulateurs, puisque la plupart des contreparties réelles (schémas, acquéreurs, opérateurs, régulateur) sont absentes. Le Lab en fournit cinq :

- **Simulateur de terminal et d'acquéreur ISO 8583** : génère des 0100/0200/0400/0800 avec profils de charge et injections d'erreurs (timeouts, messages malformés).
- **Simulateur d'émetteur** : répond aux demandes d'autorisation avec des règles paramétrables (plafonds, cartes bloquées, codes réponse).
- **DFSP simulés (Mojaloop Testing Toolkit)** : jouent le rôle de participants émetteurs et bénéficiaires, avec règles de réponse synchrones et de callback.
- **PSP simulés (connecteurs de test Hyperswitch)** : taux de succès, latence et coûts paramétrables pour tester le routage.
- **Opérateur central ISO 20022 simulé** : reçoit pacs.008, applique des contrôles, renvoie pacs.002 et camt.054 selon un guide d'implémentation fictif.

9.3 Jeux de tests positifs et négatifs

Tableau 18 — Extrait du catalogue de tests

ID	Scénario	Type	Résultat attendu
T-01	P2P 25 000 XOF entre deux wallets valides	Positif	SETTLED, soldes mis à jour
T-02	Rejeu de T-01 avec la même clé d'idempotence	Négatif	Même réponse, aucune nouvelle écriture
T-03	Montant supérieur au plafond KYC L1	Négatif	Refus OPA motivé, aucun hold
T-04	Bénéficiaire inconnu (lookup 3204)	Négatif	FAILED, hold libéré
T-05	Rail indisponible (HTTP 503) puis rétabli	Résilience	Retry automatique puis SETTLED
T-06	Rail indisponible au-delà des retries	Résilience	FAILED, compensation exécutée

ID	Scénario	Type	Résultat attendu
T-07	Rafale vers bénéficiaire récent	Fraude	ALRT, interdiction, fonds libérés
T-08	Autorisation carte puis reversal	Positif	0110/00 puis 0410/00, solde restauré
T-09	pacs.008 avec montant décimal en XOF	Négatif	pacs.002 RJCT, code motif montant
T-10	Jeton expiré ou scope insuffisant	Sécurité	401 / 403 à la passerelle, aucune trace métier
T-11	Redémarrage du worker pendant une saga	Résilience	Reprise sans doublon ni perte
T-12	Perte d'un broker Kafka	Résilience	Aucune perte d'événement (min.insync = 2)

9.4 « Casser » : chaos engineering

Le verbe « casser » est sans doute le plus formateur. Les scénarios de chaos sont déclarés en code (par exemple avec Chaos Mesh) et exécutés en campagne, avec une hypothèse, une mesure et une conclusion documentées.

Tableau 19 — Expériences de chaos engineering

Expérience	Hypothèse	Mesure	Critère de succès
Suppression du pod worker de saga	Temporal reprend les workflows sur un autre worker	Workflows terminés, doublons	0 doublon, 0 perte
Latence réseau de 800 ms vers Mojaloop	Les timeouts et retries absorbent la dégradation	Taux d'échec, latence p95	Échecs < 2 %, compensations correctes
Arrêt d'un broker Kafka	Le cluster reste disponible	Erreurs de production	Aucune perte, reprise automatique
Base du ledger en lecture seule	La saga échoue proprement	États finaux	Aucun paiement « à moitié » réglé
Expiration forcée des certificats internes	Les alertes détectent avant l'incident	Délai de détection	Alerte ≥ 7 jours avant expiration

9.5 Benchmark

Le benchmark vise la **comparabilité**, pas le record. Chaque campagne fixe l'environnement (niveau 3), les versions, le profil de charge, la durée, les seuils et publie ses résultats bruts. Les indicateurs retenus sont le débit soutenu (transactions par seconde), la latence (p50, p95, p99), le taux d'erreur, la consommation de ressources par transaction et le comportement en saturation.

perf/scenarios/instant-p2p.js — profil de charge « instantané »

```
import http from 'k6/http';
import { check } from 'k6';
export const options = {
  scenarios: { instant: { executor: 'constant-arrival-rate', rate: 600, timeUnit: '1s',
                          duration: '10m', preAllocatedVUs: 200 } },
  thresholds: { http_req_duration: ['p(95)<500'], http_req_failed: ['rate<0.005'] },
};
export default function () {
  const res = http.post(`${__ENV.API}/payments`, JSON.stringify(randomP2P()), { headers:
headers() });
  check(res, { 'status is 201': (r) => r.status === 201, 'has paymentId': (r) => !!
r.json('paymentId') });
}
```



Figure 21 — Résultat d'une campagne k6 — seuil de latence tenu, seuil d'erreur dépassé (capture illustrative)

La capture illustre un résultat typique d'une première campagne : la latence est conforme, mais le taux d'erreur dépasse le seuil. L'analyse de la trace des échecs (chapitre 11) conduit alors à identifier le goulet d'étranglement — par exemple un pool de connexions trop petit vers le ledger — puis à rejouer la campagne. C'est ce cycle mesure → analyse → correction → mesure qui constitue la valeur du benchmark.

Rigueur des chiffres

Les chiffres de performance d'un Lab dépendent fortement du matériel, des versions et de la configuration. Ils ne doivent jamais être présentés comme des capacités de production ni comparés à des chiffres publiés par des éditeurs sans reproduire strictement les mêmes conditions.

10. Sécurité et conformité

10.1 Sécurité par défaut

Tableau 20 — Contrôles de sécurité du Lab

Domaine	Contrôles appliqués dans le Lab
Identité et accès	OIDC / OAuth2 via Keycloak ; MFA pour les consoles ; comptes de service à privilèges minimaux ; RBAC Kubernetes par namespace
Réseau	NetworkPolicies deny-all par défaut ; TLS 1.2+ partout, mTLS entre partenaires et services critiques ; consoles d'administration derrière VPN
Secrets	OpenBao + External Secrets ; rotation planifiée ; aucun secret dans Git (détection automatique en CI)
Chaîne logicielle	SBOM CycloneDX, scan Trivy, signature des images (cosign), politique d'admission n'acceptant que les images signées
Données	Données synthétiques uniquement ; masquage des PAN et alias dans journaux et traces ; chiffrement des volumes
Politiques	Règles métier et d'accès en Rego, testées et revues ; journal des décisions OPA
Audit	Piste d'audit Kafka + OpenSearch ; historique Git ; journaux d'administration Keycloak et Argo CD
Tests	Scans DAST (OWASP ZAP) sur les API exposées ; revue des configurations (CIS Benchmarks Kubernetes)

10.2 PCI DSS : maîtriser le périmètre

Tout composant qui stocke, traite ou transmet des données de carte entre dans le périmètre de la norme **PCI DSS** (version 4.0.1 à la date de rédaction). Le Lab n'est pas certifié et n'a pas vocation à l'être : il n'utilise que des numéros de carte de test. En revanche, il est l'endroit idéal pour **concevoir la réduction de périmètre** avant toute trajectoire de production : tokenisation au plus tôt, isolement du coffre-fort de cartes, segmentation réseau, journalisation, gestion des clés via HSM.

10.3 Cadre réglementaire UEMOA et exigences transverses

Tableau 21 — Références réglementaires et normatives (liste non exhaustive, à valider par la conformité)

Référence	Portée pour le Lab
Règlement n°15/2002/CM/UEMOA relatif aux systèmes de paiement	Cadre général des instruments et systèmes de paiement dans l'Union ; référence pour les cas d'usage de simulation
Textes BCEAO sur la monnaie électronique et les établissements de paiement	Modélisation des acteurs (EME, EP, banques, IMF) et de leurs contraintes dans les cas d'usage
PI-SPI (BCEAO)	Cadre fonctionnel de référence pour la simulation ISO 20022 ; aucune connexion depuis le Lab
Loi uniforme relative à la lutte contre le blanchiment de capitaux et le financement du terrorisme	Exigences de surveillance et de déclaration : justification des scénarios Tazama
Lois nationales de protection des données personnelles	Justification du recours exclusif aux données synthétiques
ISO/IEC 27001	Référentiel du système de management de la sécurité de l'information appliqué au Lab

Référence	Portée pour le Lab
ISO/IEC 42001	Gouvernance des usages d'IA (assistance à la rédaction, futures règles de fraude apprenantes)

À valider

Les références réglementaires citées doivent être vérifiées dans leur version en vigueur par la direction de la conformité avant toute communication externe ou tout usage au-delà du Lab.

10.4 Conformité des licences

Tableau 22 — Synthèse des licences — à confirmer par dépôt et par version

Composant	Licence	Point d'attention pour un usage au-delà du Lab
jPOS	AGPL v3 / commerciale	Obligation de diffusion du code modifié en cas de mise à disposition réseau ; licence commerciale à envisager
Prowide ISO 20022	Apache 2.0	Fonctions avancées (validation, traduction) dans l'offre commerciale
Mojaloop	Apache 2.0	Aucune contrainte particulière ; attribution
Mifos Payment Hub EE	MPL 2.0 (dépôts vérifiés)	Copyleft au niveau fichier ; licence du moteur Zeebe à trancher (production soumise à licence depuis la 8.6)
Hyperswitch	Apache 2.0	Vérifier les modules ou services éventuellement distribués sous d'autres conditions
Formance Ledger	MIT	Vérifier le périmètre des autres modules de la plateforme Formance
Tazama	Apache 2.0	Pile de référence non destinée à la production
Keycloak, OPA, Kafka, OpenSearch, OpenTelemetry	Apache 2.0	Aucune contrainte particulière ; attribution
Temporal	MIT	Offre cloud distincte, non requise
Gravitee APIM	Apache 2.0 (édition communautaire)	Certaines fonctions relèvent de l'édition entreprise

Le Lab tient à jour un **inventaire des licences** généré automatiquement à partir des SBOM. Toute dépendance nouvelle sous licence à copyleft fort ou sous licence non open source déclenche une revue par l'OSPO ou, à défaut, par la direction juridique.

11. Observabilité et exploitation

11.1 Les signaux surveillés

Tableau 23 — Signaux surveillés

Famille	Indicateurs	Source
Signaux techniques (golden signals)	Latence, trafic, erreurs, saturation par service	Métriques OTel, Kubernetes
Indicateurs de paiement	Volume, taux de succès, taux d'autorisation, délai de règlement, montant moyen	Événements Kafka → index opp-payments-*
Intégrité financière	Équilibre du ledger, écarts de réconciliation, holds anciens non soldés	Formance, jobs de réconciliation
Risque	Alertes fraude, taux d'interdiction, faux positifs confirmés	Tazama, case management
Plateforme	Lag des consommateurs Kafka, file de tâches Temporal, état Argo CD, expiration des certificats	Exporteurs, cert-manager

11.2 Alertes et runbooks

Tableau 24 — Alertes principales et runbooks associés

Alerte	Seuil (Lab)	Première action du runbook
Taux d'échec des paiements	> 2 % sur 5 min	Ouvrir le tableau « Échecs par cause », identifier le rail, consulter les traces en erreur
Latence p95 bout en bout	> 800 ms sur 10 min	Identifier le span le plus long dans Trace analytics ; vérifier la saturation du service
Holds non soldés	> 15 min	Rechercher les workflows correspondants dans Temporal ; vérifier les compensations
Déséquilibre du ledger	Toute occurrence	Gel des tests, export des transactions, analyse immédiate (incident majeur pédagogique)
Lag consommateur Kafka	> 10 000 messages	Vérifier l'état du consommateur, augmenter les réplicas
Pic d'alertes fraude	> 3 × moyenne horaire	Vérifier une éventuelle régression de configuration des règles

11.3 Sauvegarde, restauration et réinitialisation

Trois opérations d'exploitation sont industrialisées : la **sauvegarde** quotidienne des bases (ledger, Keycloak, Hyperswitch, Temporal) et des volumes via Velero ; la **restauration** testée mensuellement (le test de restauration fait partie du catalogue de chaos) ; la **réinitialisation** complète du Lab à un état de référence, indispensable avant une session de formation ou une campagne de benchmark. La réinitialisation s'appuie sur le dépôt GitOps et le générateur de données synthétiques.

12. Grille de décision Build / Buy / Reuse / Extend / Replace

12.1 Les cinq options

Tableau 25 — Les cinq options de décision

Option	Définition	Quand la retenir
Build	Développer en interne un composant spécifique	Différenciation stratégique, absence d'offre adaptée, compétences disponibles
Buy	Acquérir une solution commerciale (licence ou service)	Fonction non différenciante, exigence de certification, besoin de support contractuel
Reuse	Utiliser une brique open source telle quelle	Brique mature, communauté active, adéquation fonctionnelle élevée
Extend	Utiliser une brique open source en l'étendant (connecteurs, plugins, contributions)	Socle solide mais manques ciblés, capacité de contribution en amont
Replace	Remplacer un composant existant par une alternative	Obsolescence, coût, dépendance excessive, non-conformité

12.2 Critères et pondération

Tableau 26 — Critères de la grille de décision (pondération indicative)

Critère	Poids	Mesuré dans le Lab par
Adéquation fonctionnelle	20 %	Couverture des cas d'usage, écarts identifiés
Maturité et communauté	15 %	Fréquence des versions, contributeurs, gouvernance, adoption
Performance et résilience	15 %	Résultats de benchmark et de chaos
Effort d'intégration	15 %	Jours-homme réels consommés dans le Lab
Sécurité et conformité	15 %	Vulnérabilités, capacités de contrôle, périmètre PCI
Licence et modèle économique	10 %	Analyse de licence, coûts de support éventuels
Compétences et souveraineté	10 %	Disponibilité locale des compétences, maîtrise du code

12.3 Exemple de lecture (illustratif)

Le tableau ci-dessous illustre la forme d'une restitution. Les recommandations sont fictives et n'ont de valeur qu'une fois les mesures réelles du Lab disponibles ; elles dépendent en outre du contexte propre à chaque institution.

Tableau 27 — Exemple de restitution de la grille (valeurs fictives)

Fonction	Brique Lab	Score (/5)	Orientation illustrative	Justification type
Switching ISO 8583	jPOS	4,1	Reuse / Buy (licence)	Maturité élevée ; question de licence AGPL à trancher
Messages ISO 20022	Prowide	4,0	Extend	Bibliothèque solide ; règles de marché à construire

Fonction	Brique Lab	Score (/5)	Orientation illustrative	Justification type
Paiement instantané	Mojaloop	3,6	Reuse (simulation)	Excellent support pédagogique ; pas d'usage de production ciblé
Hub d'intégration	Payment Hub EE	3,4	Extend	Connecteurs à développer ; licence Zeebe à arbitrer
Orchestration	Hyperswitch	3,8	Extend	Connecteurs rails africains à contribuer
Ledger	Formance	4,2	Reuse	Modèle clair, idempotence native
Fraude / LBC-FT	Tazama	3,5	Extend	Règles et calibrage à construire avec la conformité
IAM	Keycloak	4,5	Reuse	Standard de fait, compétences disponibles
Streaming	Kafka	4,6	Reuse	Standard de fait
Observabilité	OTel + OpenSearch	4,3	Reuse	Standard ouvert, souveraineté des données

12.4 Modèle de fiche de décision (ADR)

Modèle d'Architecture Decision Record

```
# ADR-0007 — Moteur de ledger pour la plateforme de paiement
Statut   : proposé | accepté | remplacé par ADR-00xx
Date     : 2026-09-xx
Contexte : besoin d'un ledger à double entrée, idempotent, auditable, multi-devises.
Options  : Build (PostgreSQL + code maison) | Reuse (Formance) | Buy (éditeur X)
Mesures  : benchmark Lab (tps, p95), effort d'intégration (j/h), résultats de chaos,
           analyse de licence, disponibilité des compétences.
Décision : ...
Conséquences : positives / négatives / risques / actions de suivi.
```

13. Gouvernance, équipe et feuille de route

13.1 Organisation

Tableau 28 — Organisation type d'un Lab d'équipe

Rôle	Responsabilités	Charge indicative
Sponsor	Porte l'initiative, arbitre les priorités, valorise les résultats auprès de la direction	Ponctuelle
Responsable du Lab (Lab owner)	Pilotage, feuille de route, qualité des livrables, animation de la communauté	50 %
Architecte référent	Architecture cible, contrats, ADR, grille de décision	50 %
Ingénieurs Platform / SRE (2)	Socle, GitOps, observabilité, sécurité, chaos	100 %
Ingénieurs Software (3)	Adaptateurs, saga, connecteurs, simulateurs, tests	100 %
Correspondants risques / conformité / sécurité	Scénarios de fraude, revue des contrôles, validation des références	10 à 20 %

13.2 Feuille de route sur douze semaines

Tableau 29 — Feuille de route indicative

Semaines	Jalon	Livrables
S1 – S2	Socle et services transverses	Cluster, GitOps, secrets, Keycloak, OPA, Gravitee, Kafka, observabilité
S3 – S4	Ledger et switch	Formance, plan de comptes, jPOS, cas d'usage 2
S5 – S6	ISO 20022 et saga	Adaptateur Prowide, Temporal, worker de saga, cas d'usage 4
S7 – S8	Instantané et intégration	Mojaloop, TTK, Payment Hub EE, cas d'usages 1 et 3
S9	Orchestration	Hyperswitch, connecteurs Lab, routage
S10	Fraude	Tazama, relais Kafka, cas d'usage 5
S11	Tests, chaos, benchmark	Campagnes, rapports, corrections
S12	Décision et restitution	Grille Build / Buy / Reuse / Extend / Replace, ADR, démonstration à la direction

13.3 Risques et parades

Tableau 30 — Principaux risques

Risque	Probabilité	Impact	Parade
Sous-estimation des ressources (Mojaloop, OpenSearch)	Élevée	Moyen	Dimensionnement progressif ; niveau 1 par sous-ensembles
Dérive vers un « pilote de production » non maîtrisé	Moyenne	Élevé	Charte du Lab, données synthétiques, séparation stricte des environnements
Mauvaise lecture des licences	Moyenne	Élevé	Inventaire SBOM, revue OSPO / juridique

Risque	Probabilité	Impact	Parade
Dépendance à quelques experts	Élevée	Moyen	Binômage, documentation, sessions de restitution
Connectivité et accès aux registres	Moyenne	Moyen	Miroir local des images et dépôts
Chiffres de benchmark mal interprétés	Moyenne	Moyen	Protocole publié, résultats bruts, avertissements systématiques

13.4 Indicateurs de réussite du Lab

- Nombre d'ingénieurs ayant réalisé le parcours complet (installer → benchmarker) et validé les sept cas d'usage.
- Temps de reconstruction complète du Lab depuis le dépôt (objectif : moins d'une journée).
- Nombre d'ADR produits et de décisions Build / Buy / Reuse / Extend / Replace effectivement éclairées par le Lab.
- Nombre de contributions en amont (correctifs, connecteurs, documentation) aux projets open source utilisés.
- Nombre de démonstrations réalisées auprès des métiers, des directions et des partenaires de l'écosystème.

Annexes

Annexe A — Glossaire

Tableau 31 — Glossaire

Terme	Définition
ADR	Architecture Decision Record : trace écrite, datée et argumentée d'une décision d'architecture
BPMN	Business Process Model and Notation : notation standard des processus, exécutable par Zeebe
CloudEvents	Spécification CNCF décrivant l'enveloppe standard d'un événement
DFSP	Digital Financial Service Provider : participant d'un système de paiement instantané (terme Mojaloop)
FSPIOP	API d'interopérabilité entre fournisseurs de services financiers utilisée par Mojaloop
GitOps	Pratique où l'état désiré de l'infrastructure est décrit dans Git et appliqué automatiquement
Idempotence	Propriété d'une opération qui produit le même résultat qu'elle soit exécutée une ou plusieurs fois
ISO 8583	Norme de messages pour les transactions initiées par carte
ISO 20022	Norme universelle de messagerie financière (pain, pacs, camt…)
KRaft	Mode de fonctionnement de Kafka sans ZooKeeper
Ledger	Registre comptable des mouvements et soldes, à double entrée
MTI	Message Type Indicator : type de message ISO 8583 (0100, 0110, 0400…)
NDC	Net Debit Cap : plafond de position débitrice nette d'un participant
OTLP	OpenTelemetry Protocol : protocole de transport des traces, métriques et journaux
PI-SPI	Plateforme Interopérable du Système de Paiement Instantané de l'UEMOA, opérée par la BCEAO
Rego	Langage déclaratif de politiques d'Open Policy Agent
Saga	Patron de transaction longue composée d'étapes locales et de compensations
SBOM	Software Bill of Materials : inventaire des composants d'un logiciel
STAN	System Trace Audit Number : numéro de trace ISO 8583 (champ 11)

Annexe B — Check-list de mise en service du Lab

Tableau 32 — Check-list de mise en service

#	Contrôle	Fait
1	Charte du Lab signée (données synthétiques, non-production, périmètre)	☐
2	Dépôts GitOps et applicatif créés, protections de branches actives	☐
3	Socle Kubernetes, ingress, cert-manager, Argo CD opérationnels	☐
4	OpenBao initialisé, clés de descellement séparées et conservées	☐
5	Realm Keycloak importé depuis Git ; MFA actif sur les consoles	☐
6	NetworkPolicies deny-all actives et flux autorisés testés	☐

#	Contrôle	Fait
7	Chaîne d'observabilité : trace de test visible de bout en bout	☐
8	Générateur de données synthétiques exécuté et vérifié	☐
9	Les onze phases de déploiement validées et prouvées	☐
10	Les sept cas d'usage automatisés et verts	☐
11	Première campagne de benchmark et de chaos documentée	☐
12	Inventaire des licences généré et revu	☐

Annexe C — Références

- Documentation officielle de chaque projet : jPOS, Prowide ISO 20022, Mojaloop, Mifos Payment Hub EE, Hyperswitch, Formance, Tazama, Keycloak, Apache Kafka, OpenTelemetry, OpenSearch, Gravitee, Open Policy Agent, Temporal.
- BCEAO — communiqués relatifs au lancement (30 septembre 2025) et à la connexion des participants à la PI-SPI ; site pispi.bceao.int.
- Camunda — documentation sur les licences de Camunda 8 Self-Managed (impact sur Zeebe à partir de la version 8.6).
- Spécifications : ISO 8583, ISO 20022 (catalogue des messages), CloudEvents 1.0, OpenTelemetry, OpenAPI 3, AsyncAPI, RFC 9457 (Problem Details).
- PCI Security Standards Council — PCI DSS v4.0.1.

Mention

« L'open source n'est pas une fin en soi. C'est un accélérateur d'apprentissage, de souveraineté et d'innovation. » — Lionel SAKITI

Document assisté par IA, gouverné par ISO/IEC 42001. Relecture humaine et validation technique requises avant diffusion.