# PROOF OF CONCEPT REPORT

## Core Digital Transformation for Retail Banking

*Microservice Deployment Feasibility & Data Synchronization Validation*

| Date | Prepared By | Version |
|---|---|---|
| February 18, 2026 | Systems Architect / Data Engineering Lead | 1.0 — Initial Release |

## Task 1: Define PoC Scope and Objective

| Section | Description |
|---|---|
| **PoC Objective** | Validate microservice deployment feasibility by testing an event processing pipeline using Apache Kafka and data synchronization between PostgreSQL and IBM Cloud Object Storage. |
| **Scope (In)** | Containerized Spring Boot microservice deployment via Docker; Kafka topic creation and message publishing; PostgreSQL event_log table writes; REST API endpoint validation via Postman; Cloud console connectivity check. |
| **Scope (Out)** | Production-level data volumes; live PCI-DSS cardholder environments; full OAuth 2.0 implementation; multi-region failover testing; mobile app integration. |
| **Success Criteria** | API response latency < 2 seconds; Kafka message delivery confirmed within 150ms; 100% message delivery (no loss); PostgreSQL writes confirmed; container boot time < 60 seconds; all API endpoints return expected HTTP status codes. |
| **Selected Use Cases** | 1. Real-time fraud detection event flow 2. API integration with external audit bureau (Credit Bureau API) 3. Containerized microservice deployment (Auth, Loan, Onboarding) 4. Data synchronization between PostgreSQL and cloud object storage |
| **Assumptions** | Sandbox environment with sample datasets only; mock API keys used for third-party credit bureau integration; Docker Desktop available on local dev machines; IBM Cloud or AWS free-tier access provisioned; 100 events/sec simulated load (not production-scale). |

## Task 2: PoC Environment and Tools Used

| Component | Technology | Version | Setup Method | Notes |
|---|---|---|---|---|
| Microservice | Spring Boot | 3.2.x | Containerized via Docker | Runs on port 8080 |
| Event Broker | Apache Kafka | 3.6.x | Docker Compose | Topic: fraud-events |
| Database | PostgreSQL | 15.x | Docker container | Table: event_log |

| | | | | |
|---|---|---|---|---|
| API Testing | Postman | 10.x | Local installation | REST endpoint validation |
| Container Runtime | Docker Desktop | 25.x | Local / IBM Cloud | Kubernetes-ready images |
| Cloud Console | IBM Cloud / AWS Free Tier | N/A | Browser-based provisioning | Object storage & registry |
| Monitoring | Kafka CLI | 3.6.x | Command line tools | Consumer lag tracking |

## Task 3: Document Test Scenarios and Outcomes

| Test ID | Scenario | Expected Outcome | Actual Outcome | Pass/Fail |
|---|---|---|---|---|
| TC-01 | Publish 100 Kafka fraud-events and validate consumer processing latency | Events logged in event_log table within 150ms; zero consumer errors | Avg. latency: 120ms; all 100 events logged; zero consumer errors | **PASS** |
| TC-02 | Restart Kafka microservice consumer mid-stream to test auto-recovery | Consumer reconnects within 30 seconds; no message loss observed | Reconnected in 22 seconds; 0 messages lost; offset resume confirmed | **PASS** |
| TC-03 | API call to mock Credit Bureau endpoint to validate integration response | HTTP 200 response with valid credit score JSON payload within 2 seconds | HTTP 200 returned; response time 1.3s; JSON payload correctly parsed | **PASS** |
| TC-04 | Simulate PostgreSQL container failure (DB failover scenario) | Retry logic activates within 5 seconds; service does not crash; alert logged | Retry kicked in after 1 attempt (4.2s); service remained live; alert generated | **PASS** |
| TC-05 | Simulate network latency (500ms artificial delay) on Kafka topic publish | System continues processing; latency spikes logged; no data loss | All 100 events still delivered; latency spike logged as 620ms; no loss | **PASS** |

## Task 4: Capture Findings, Limitations, and Recommendations

### 4.1 Findings

| Category | Finding | Impact | Recommendation |
|---|---|---|---|
| Performance | Kafka latency averaged 120ms against a 150ms threshold — within target range | Meets goal | Scale Kafka partitions from 1 to 3 before production load testing to maintain margin |

| | | | |
|---|---|---|---|
| Integration | Credit Bureau mock API responded correctly; real OAuth 2.0 not yet tested | Good (partial) | Add OAuth 2.0 bearer token flow before production; implement retry with exponential backoff |
| Security | Basic auth only used in sandbox; no TLS between services; RBAC not yet enforced | Partial | Implement mTLS for service-to-service communication; integrate Kubernetes RBAC before staging promotion |
| Deployment | Spring Boot containerized successfully via Docker; image size ~145MB | Good | Integrate CI/CD pipeline next; add automated vulnerability scanning (Trivy) before registry push |
| Data Sync | PostgreSQL write latency within acceptable range; cloud object storage sync not fully tested | Partial | Complete object storage sync testing in next PoC iteration with AWS S3 or IBM COS |

### 4.2 Limitations

- Sample dataset of 100 events used — not representative of production volumes (target: 10,000+ concurrent users)
- No failover environment tested — single-node Kafka and PostgreSQL only; no multi-AZ setup validated
- Mock API keys used for Credit Bureau — live API authentication, rate limits, and SLAs not yet validated
- No production-level security controls tested (TLS, RBAC, MFA, secrets management)
- Cloud object storage synchronization (IBM COS / AWS S3) not fully validated — local PostgreSQL only

### 4.3 Improvement Points

- Scale Kafka to 3+ partitions and conduct load testing at 1,000 events/sec to validate production throughput
- Integrate CI/CD pipeline (GitHub Actions / IBM Toolchain) to automate build, scan, and deployment
- Introduce mTLS and Kubernetes NetworkPolicies before staging environment promotion
- Complete OAuth 2.0 integration with the real Credit Bureau API endpoint
- Conduct disaster recovery drill — simulate full Kafka broker failure and validate consumer group rebalancing

## Task 5: Evaluate PoC Against Success Criteria

| Success Criteria | Result | Met | Notes |
|---|---|---|---|
| API response latency < 2 seconds | 1.3 seconds (avg) | **YES** | Well within threshold; production load test recommended |
| Kafka message delivery within 150ms | 120ms (avg) | **YES** | 30ms headroom; scale partitions before production |
| 100% message delivery — no message loss | 100/100 delivered | **YES** | Validated across 5 test runs including restart scenario |
| PostgreSQL writes confirmed for all events | All 100 records written | **YES** | event_log table verified post each test run |

| | | | |
|---|---|---|---|
| Container boot time < 60 seconds | 38 seconds (avg) | **YES** | Docker image optimized; Kubernetes liveness probe configured |
| All API endpoints return expected HTTP status codes | 200 / 201 returned | **YES** | All 5 endpoints validated via Postman collection |
| Cloud object storage synchronization validated | Partially tested | **PARTIAL** | PostgreSQL sync confirmed; IBM COS/S3 deferred to next iteration |

## Task 6: Validate Your PoC Report

| Validation Item | Yes / No |
|---|---|
| Objectives defined clearly with measurable success criteria | **YES** |
| Scope documented — in scope and out of scope items identified | **YES** |
| Prototype built — containerized microservice deployed and connected to Kafka and PostgreSQL | **YES** |
| Test scenarios completed — 5 test cases executed with documented outcomes | **YES** |
| Findings documented — performance, integration, security, and deployment categories covered | **YES** |
| Success criteria mapped — all 7 criteria evaluated with results and notes | **YES** |
| Go / No-Go decision included — see recommendation below | **YES** |
| Documentation complete — all tasks populated with relevant content | **YES** |

## Go / No-Go Recommendation

| | |
|---|---|
| **CONDITIONAL GO** | 6 of 7 success criteria fully met. The PoC demonstrates that the core microservice architecture, Kafka event pipeline, and PostgreSQL integration are viable for the Core Digital Transformation initiative. Conditional approval is granted subject to: (1) completing cloud object storage synchronization testing, (2) implementing mTLS and OAuth 2.0 before staging promotion, and (3) validating performance at production-scale load (10,000+ concurrent users). |