Payment Fraud Check Demo (PoC)

Integration of PPS, BS, and FCS with Messaging & REST

Ankur Kapila

Agenda

- Problem Statement & Objectives
- Systems in Scope
- Solution
- Demo Flow
- Technology Stack & Trade-offs
- Deployment & Observability
- Q&A

Problem Statement

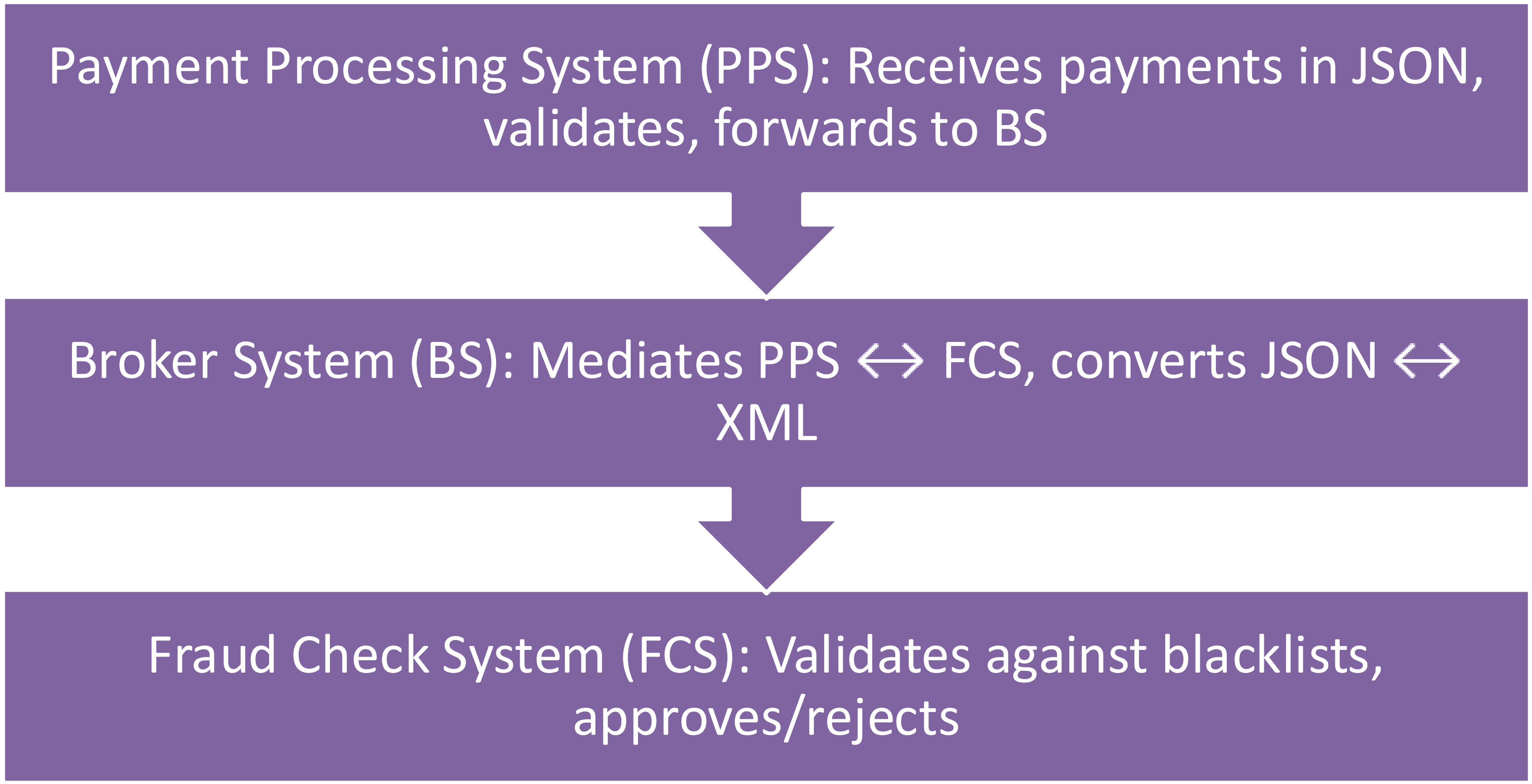
- BANK REQUIRES FRAUD CHECKS BEFORE HONORING PAYMENTS

- ENSURE TRACEABILITY, VALIDATIONS, AND AUDIT LOGS

- TWO INTEGRATION APPROACHES: MESSAGING AND REST

- REUSABLE, DEMONSTRABLE WITHIN 90 MINUTES

Systems in Scope

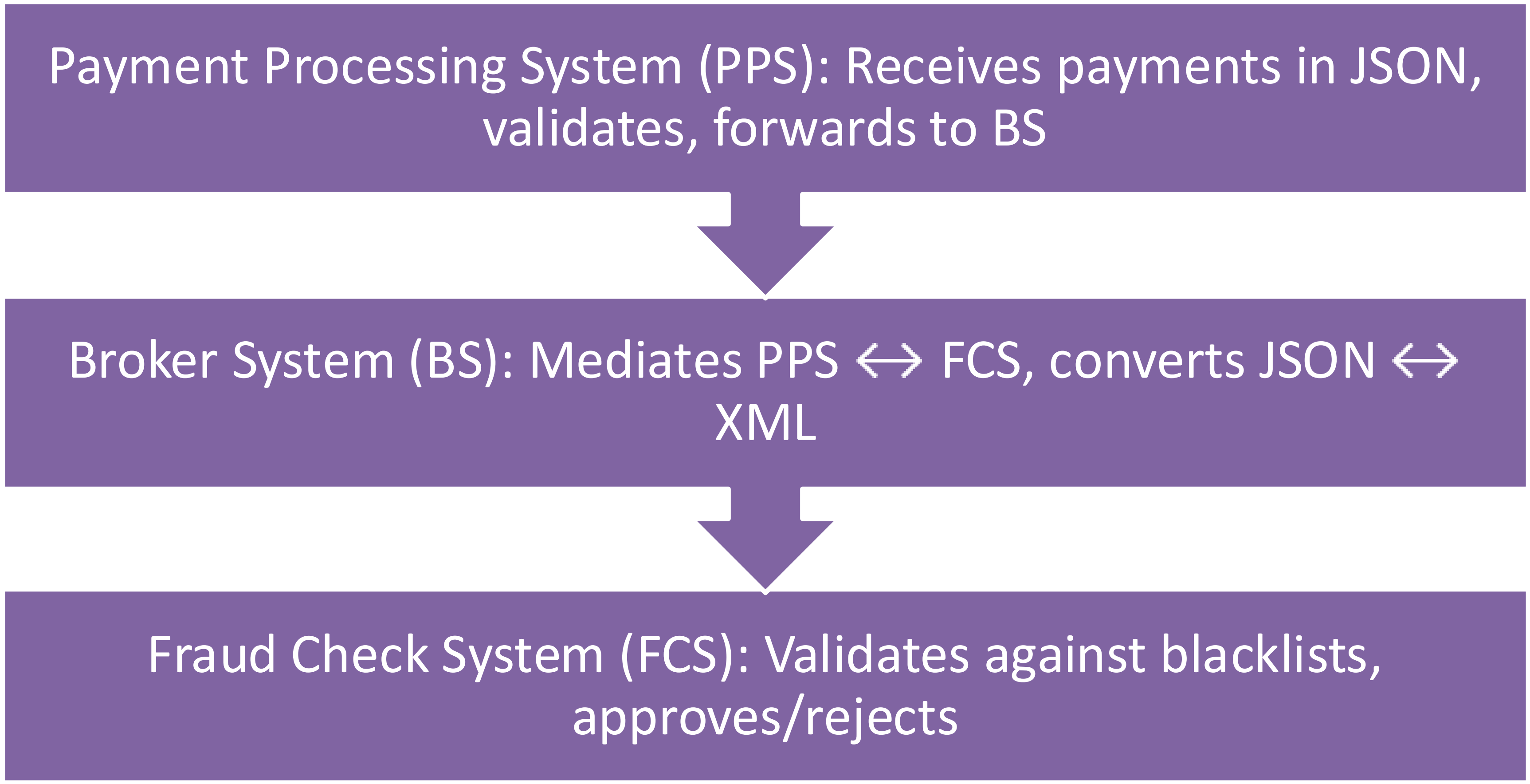


Payment Processing System (PPS): Receives payments in JSON, validates, forwards to BS

Broker System (BS): Mediates PPS ↔ FCS, converts JSON ↔ XML

Fraud Check System (FCS): Validates against blacklists, approves/rejects

Assump

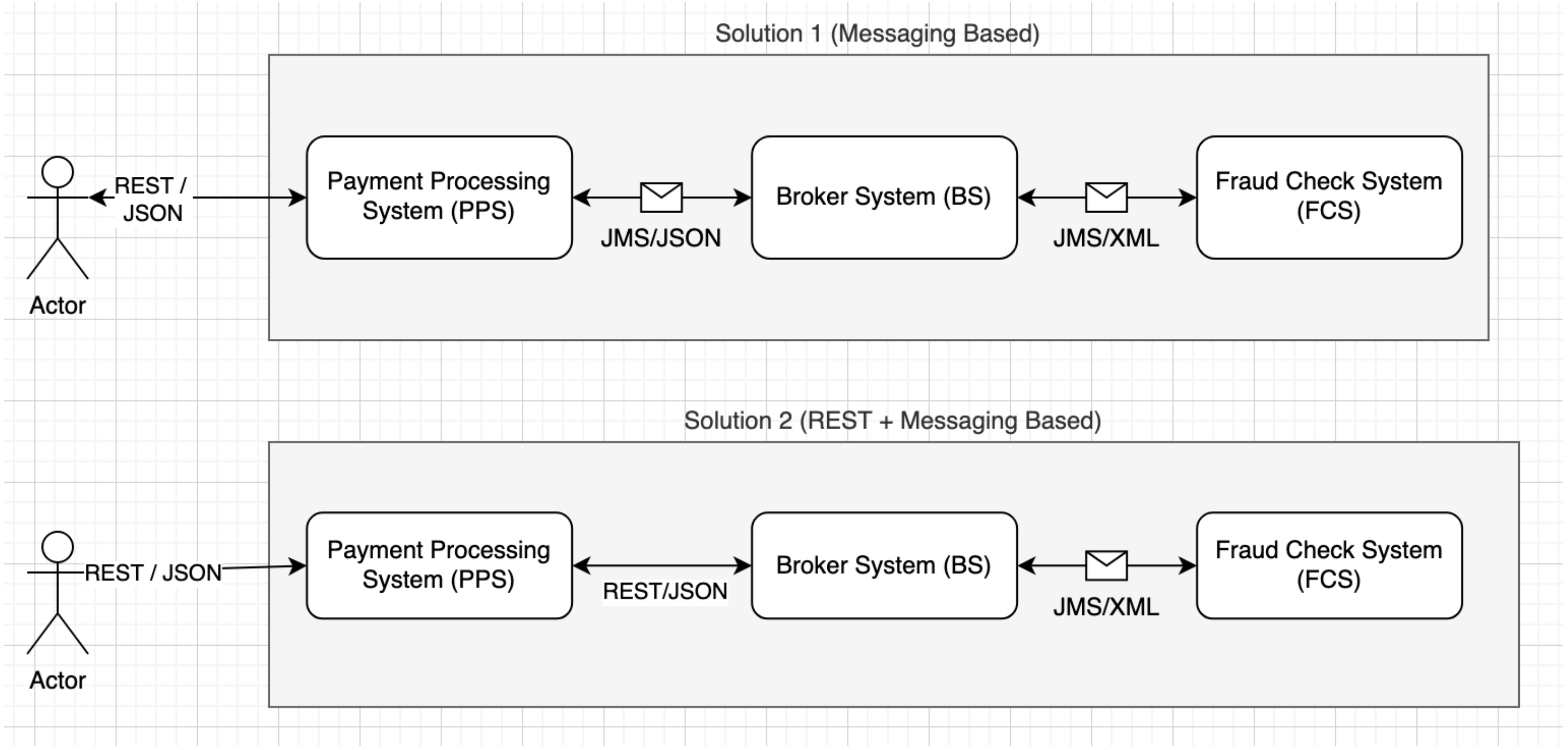


Payment Processing System (PPS): Receives payments in JSON, validates, forwards to BS

Broker System (BS): Mediates PPS ↔ FCS, converts JSON ↔ XML

Fraud Check System (FCS): Validates against blacklists, approves/rejects

Demo Flow


Solution 1 (Messaging Based)
Actor
REST / JSON
Payment Processing System (PPS)
JMS/JSON
Broker System (BS)
JMS/XML
Fraud Check System (FCS)
Solution 2 (REST + Messaging Based)
Actor
REST / JSON
Payment Processing System (PPS)
REST/JSON
Broker System (BS)
JMS/XML
Fraud Check System (FCS)

Technology Stack

- Spring Boot / Java 21
- Apache Camel
- ActiveMQ Artemis (JMS provider)
- JSON & XML (Jackson + JAXB)
- Spring / Jakarta Validation
- Docker Container/Docker Compose
- Observability
  - Open Telemetry (metrics/traces/logs)
  - Grafana LGTM (Loki, Grafana, Temo, Mirmir)

Technology Selection – Message Queue

Options Considered

- **ActiveMQ / ActiveMQ Artemis**
  Open-source JMS provider, widely adopted, easy integration with Java and Apache Camel.
- **RabbitMQ**
  AMQP-based, strong ecosystem, popular in microservices and event-driven architectures.
- **Kafka**
  Distributed log streaming platform, strong for large-scale event streaming, but higher setup and operational overhead.
- **Azure Service Bus / AWS SQS** (managed cloud options)
  Reliable, but tied to cloud providers; not ideal for a vendor-neutral PoC.
- **IBM MQ**
  Enterprise-grade, but heavyweight for a demo and not open-source.

ActiveMQ Artemis

- Native JMS support (fits directly with requirement: “use Apache AMQ (or any other open source JMS provider)”

Technology Selection – Observability

Options Considered

- **Grafana + Prometheus + Loki**
  Most widely used open-source stack for metrics, logs, visualization.
- **Grafana Cloud (Free tier)**
  Managed option with hosted dashboards and log ingestion.
- **ELK Stack (Elasticsearch, Logstash, Kibana)**
  Powerful, but heavier footprint, more effort to set up.
- **OpenTelemetry + Jaeger**
  Strong for tracing, can be integrated into a larger observability setup.
- **OpenShift built-in monitoring (Prometheus, EFK stack)**
  Nice-to-have option if deployed on OpenShift.
- **SigNoz / OpenObserve**
  Newer open-source observability platforms with tracing + metrics + logs built-in, simpler setup than ELK.
- **Paid APM** like data dog, new relic and applications insights etc.

Final Choice

- **Grafana** grafana/otel-lgtm
  - Works well in both local demo and cloud environments.
  - Batteries included, easy switch to cloud gtafana free trial
  - Easier to demo flows visually with dashboards.
  - **Easy to leverage Grafana Cloud free trial**, reducing setup effort.

Technology Selection – Grafana OTEL LGTM

Component	**Purpose / What it Provides**
OpenTelemetry Collector	Receives telemetry (metrics, logs, traces) via OTLP (gRPC & HTTP) on default ports (4317, 4318). It acts as data ingestion/processing agent. (Grafana Labs)
Grafana	UI frontend to view dashboards, explore metrics/traces/logs etc. Comes pre-configured with data sources pointing to the other bundled components. (Grafana Labs)
Prometheus	Stores metrics; used for metrics ingestion and querying via PromQL. (Grafana Labs)
Loki	Logs backend; ingestion and querying of logs via LogQL etc. (Grafana Labs)
Tempo	Tracing backend; stores and allows exploring traces collected via OpenTelemetry. (Grafana Labs)
Pyroscope	For continuous profiling (i.e. observing program execution, CPU/memory usage, etc over time) – bundled for profiling data. (GitHub)

Technology Selection – Observability

- **Unified, vendor-neutral standards**
  - **OpenTelemetry (OTel)** — spec + SDKs + Collector for **traces, metrics, logs (and profiling)** with **OTLP** as the wire protocol; auto-instrumentation for many languages; exports to many backends (Grafana Tempo/Loki/Prometheus, Elastic, Datadog, New Relic, etc.).
- **Legacy/open standards (trace-centric)**
  - **OpenTracing** — tracing API (now merged into OTel; maintenance only).
  - **OpenCensus** — Google-origin metrics/tracing (merged into OTel; maintenance only).
  - **Zipkin/Brave** — classic tracing libs + Zipkin server; great for traces, not unified for metrics/logs.
- **Backend-specific client libraries/agents**
  - **Prometheus client libs** — excellent for **metrics** only; needs a separate story for traces/logs.
  - **Jaeger client libs** — tracing only; superseded by OTel in most new builds.
  - **Elastic APM, Datadog, New Relic, Dynatrace agents** — easy end-to-end experience, but **vendor lock-in**, mixed coverage across languages, and switching backends later is costly.

OpenTelemetry

- OpenTelemetry for unified, vendor-neutral telemetry with the flexibility of the Collector and OTLP, perfect fit for our Grafana LGTM demo.
- Didn’t pick: Legacy APIs (OpenTracing/OpenCensus), single-signal clients (Prometheus-only, Jaeger-only), or vendor agents—either limited scope or introduce lock-in.

Next Steps

- Live demo of both solutions
- Code walkthrough (reusable components)
- Q&A