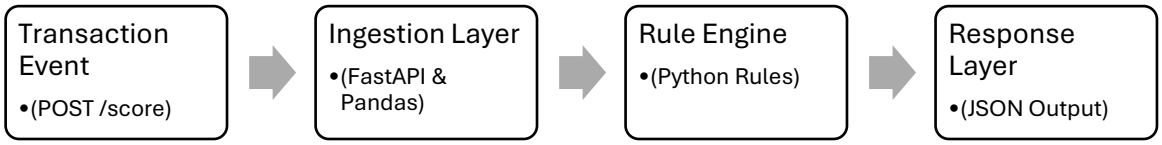
**CloudWalk Technical Case – Anti-Fraud Solution**

**Data Analyst - Risk Analyst I**

**Daniel Sousa**

**Architecture Diagram**




---

**Layer Descriptions & Technical Justification**

1. **Ingestion Layer**
    - **Components:** FastAPI endpoint (/score) and Pandas DataFrame preloaded from CSV.
    - **Responsibility:** Accepts incoming JSON transaction payloads, normalizes the timestamp field, and prepares data for rule evaluation.
    - **Justification:** FastAPI ensures low-latency HTTP handling; Pandas provides efficient time-based filtering on historical data.
2. **Rule Engine**
    - **Components:** Two pure-Python functions - **rule_velocity** and **rule_low_value_tests** - applied in sequence.
    - **Responsibility:**
        - **Velocity Rule:** Flags if a **user_id** has more than three transactions in the preceding 10 minutes.
        - **Low-Value Test Rule:** Flags if a **device_id** has more than two transactions under R$ 10 in the preceding 10 minutes.
    - **Justification:** Deterministic rules are transparent, easy to tune.
3. **Response Layer**
    - **Components:** Logic that aggregates rules results in **rule_flags** and decides decisive **action** ("**hold_for_review**" or "**approve**").
    - **Responsibility:** Returns a concise JSON object showing which rules fired and the resulting decision.
    - **Justification:** Clear, actionable output format supports downstream integration with dashboards or manual review processes.

---

**Prototype Code Snippets**

```
@app.post("/score")
```

```
def score_transaction(txn: dict):
    # Normalize timestamp
    txn_time = txn.get("transaction_date") or txn.get("transaction_time")
    txn = {**txn, "transaction_date": txn_time}

    # Evaluate each rule
    flags = {
      f"rule_{i+1}": rule(txn)
      for i, rule in enumerate(rules)
    }

    # Final decision
    action = "hold_for_review" if any(flags.values()) else "approve"

    return {
      "action": action,
      "rule_flags": flags
    }
```

- **Normalization:** Ensures the service accepts either **transaction_date** or **transaction_time**.
- **Rule Evaluation:** Iterates over **rules** list, applies each function, and collects Boolean results.
- **Decision Logic:** If any rule is **True**, the transaction is flagged for review; otherwise, it is approved.

---

**Evolution Plan**

1. **Monitoring & Metrics**
    - Instrument counts of approved vs. held transactions; track average evaluation latency.
    - Alert unusual spikes in **hold_for_review** rates.
2. **Rule Management Interface**
    - Build a lightweight GUI or CLI to add, remove, and tune rules without code redeployment.
3. **Feature Enrichment**

- Integrate more data sources (e.g., IP geolocation, device fingerprint scores) into rule conditions.

4. **Optional ML Augmentation**
    - When comfortable, append a machine-learning model that scores edge-case transactions; keep hybrid decision logic (rules + ML).
5. **Automated Testing & CI/CD**
    - Write unit tests for each rule function.
    - Deploy via CI pipeline to run tests and linting on each commit.