Stratégie d'intégration du ML

1) Objectifs & exigences

- **Objectif principal** : fournir scoring ML en temps réel (fraude, risk, recommandations) et analyses offline robustes (retraining, KPI business).
- **Contraintes** : latence d'inférence ≤ 100 ms (fraude), exactly-once ingestion des features critiques, traçabilité complète (reproductibilité).
- **SLA** : disponibilité du service de scoring ≥ 99.95%, délai de mise à jour des features critiques < 200 ms depuis événement source.

---

2) Topologie Feature Store

Offline Feature Store (historique)

- **Stockage** : S3 / Data Lake (format Parquet, partitionné par date, merchant_id).
- **Usage** : entraînement, backfills, audits, lineage.
- **Schema** : entity_id, feature_name_1, ..., as_of_timestamp, version.

Exemple de table offline (parquet)

customer_id	as_of	avg_tx_7d	cnt_tx_30d	chargeback_rate_90d	version
uuid-1	2025-11-17T00:00Z	12.5	10	0.02	v3

Online Feature Store (low-latency)

- **Stockage** : Redis (hashes) ou DynamoDB (PK: entity_id)
- **Usage** : lookup en < 50ms lors de scoring temps réel.
- **Population** : stream processing (Flink/Spark) consomme Kafka → compute features → write to Online FS.

Pattern : write-through via CDC + streaming jobs → Online FS; periodic reconciliation via batch job (Airflow) pour garantir cohérence.

---

3) Pipeline d'entraînement (offline)

Étapes

1. **Extraction** : read from Offline FS (S3 Parquet), DWH facts, NoSQL aggregated features.

2. **Feature engineering** : windowed aggregations, joins, normalization (Spark/DBT).
3. **Train / Validate** : train models (XGBoost/LightGBM / OSS deep nets), k-fold, holdout set.
4. **Evaluation** : metrics (AUC, precision@k, recall @ given FPR).
5. **Packaging** : save model artifact + preprocessing pipeline + feature list.
6. **Register** : push to Model Registry (MLflow) with metadata (training data snapshot, code commit, hyperparams).
7. **Approval** : automated tests (smoke tests + policy checks) → mark as "staging"/"production".

Artefacts stockés

- model.pkl / model.tar.gz
- preprocessing script / scaler config
- feature spec JSON (names, types, as_of semantics)
- training dataset hash (for reproducibility)

---

4) Pipeline d'inférence (online)

Mode synchrone (realtime scoring)

Flow : event arrives → get entity_id → lookup features in Online FS → call scoring API (gRPC/HTTP) → combine rule engine → response (accept / review / block).

Exigences latence :

- Feature lookup: ≤ 20 ms
- Model inference: ≤ 50 ms (quantized/optimized model)
- Orchestration & network: ≤ 30 ms
  Total target ≤ 100 ms

Mode asynchrone (batch scoring)

Used for nightly rescoring, cohort scoring, offline experiments.

---

5) Exemple concret : schéma features & ingestion pseudo-code

JSON spec d'un enregistrement de features (online)

```
{
  "customer_id": "uuid-1",
  "as_of": "2025-11-18T10:00:00Z",
  "avg_tx_7d": 12.5,
  "cnt_tx_30d": 24,
  "velocity_1h": 5,
  "device_risk": 0.41,
  "last_tx_amount": 1200,
  "version": "v3"
}
```

Pseudo-code ingestion stream (Flink-like)

```
for event in kafka.consume('transactions'):
    key = event.customer_id
    # update sliding windows
    features = window_aggregator.update_and_get(key, event)
    redis.hset(key, mapping=features)  # write to online FS
    sink_parquet.append(features, partition=event.date)
```

---

6) Model serving & déploiement

Options de serving

- **Triton / KFServing** for low-latency GPU/CPU serving
- **FastAPI + Uvicorn + Gunicorn** for small models
- **Serverless (Lambda)** for extremely bursty but stateless scoring (watch cold starts)

API contract (gRPC / HTTP)

Endpoint: POST /score
Payload:

```
{
 "entity_id": "uuid-1",
 "features": null  // optional: if null server will fetch from online FS
}
```

Response:

```
{
 "score": 0.97,
 "model_version": "fraud_v8",
 "explanations": {"velocity_1h": 0.5, "device_risk": 0.3}
}
```

Canary & rollout

- Canary on 1% traffic → 5% → 25% → 100%
- Auto rollback if key metric degrade beyond threshold (AUC drop, FP increase, latency spike).

---

7) Versioning et reproducibility

- **Model Registry** : MLflow storing artifact + metadata (git commit, dataset hash, params).
- **Feature spec versioning** : every feature has version and computation logic version (feature_v3).
- **Data snapshot** : store snapshot id (S3 path) used for training in registry.

---

8) Monitoring, alerting & drift detection

Metrics à collecter

- **Model perf** : AUC, precision@k, recall, FPR, FNR (daily/weekly)
- **Business KPIs** : detected frauds prevented, revenue impact
- **Inference latency** : p50/p95/p99
- **Feature freshness** : last_update_time per key (online FS)
- **Input distribution** : histogram of key features (mean/std), compare baseline

Drift detection

- **Statistical tests** : KL divergence, PSI (Population Stability Index) between training and live distributions
- **Alerts** : if PSI > threshold (e.g. 0.2) → ticket + pause auto-rollouts

Tools

- Prometheus + Grafana for metrics
- ELK/Splunk for logs & explainability traces
- Evidently.ai or custom jobs for drift

---

9) Logging & explainability

- **Per-inference log** : entity_id, features vector (hashed or masked), model_version, score, latency, decision. Stocker en NoSQL (fraud_events) ou log stream vers SIEM.
- **Explainability** : SHAP/TreeSHAP for offline; lightweight feature contributions for online (top-5 contributors).

---

10) Tests & validation avant prod

Tests automatiques

- **Unit tests** : feature computations
- **Integration tests** : end-to-end pipeline ingest→online FS→serve
- **Performance tests** : load test scoring endpoint (target QPS)
- **Backtest** : train on period T1, test on T2, run replay on historical data to estimate impact.

Operational checks

- Data quality (null rate, cardinality) thresholds
- Feature drift checks
- Schema validation (Avro/Protobuf) on Kafka topics

---

11) Sécurité & conformité ML

- **No storage of raw PAN** in features; mask / tokenize sensitive fields before storing features.
- **Access control** : only authorized roles can read raw features; analysts get masked views.
- **Explainability logs** : PII removed before persisting for compliance.
- **Reproducibility for audits** : store training data snapshot, feature spec, model artifact.

---

12) Plan de mise en production & rollback

Déploiement recommandé

1. Staging environment: full pipeline with sampled traffic (10%)
2. Canary: deploy model to 1% production traffic
3. Monitor metrics (7–14 days)
4. Gradual ramp-up to 100% with automated checks

Rollback criteria

- latency p95 > SLA
- AUC decrease > configured delta
- spike in false positives/false negatives beyond threshold

Rollback procedure

- Mark model as deprecated in registry
- Route traffic back to previous stable model via service mesh (envoy routing)
- Trigger re-training investigation

---

13) Exemples concrets (SQL / pseudo-queries)

a) Extraire features offline pour training (Spark SQL)

```
SELECT customer_id,
    SUM(amount)/7 as avg_tx_7d,
    COUNT(*) FILTER (WHERE created_at >= date_sub(current_date, 30)) as cnt_tx_30d,
    SUM(case when is_chargeback then 1 else 0 end)/NULLIF(COUNT(*),0) as chargeback_rate_90d,
    MAX(created_at) as as_of
FROM olap.fact_transactions
WHERE created_at >= date_sub(current_date, 120)
GROUP BY customer_id;
```

b) Check freshness (SQL)

```
SELECT customer_id, MAX(as_of) as last_feature_update
FROM ml_offline_features
GROUP BY customer_id
HAVING MAX(as_of) < date_sub(current_date, 1); -- stale > 1 day
```

---

14) KPIs à suivre (exemples)

- **Model performance** : AUC >= 0.92 (baseline)
- **Business** : reduction of chargeback rate by X%
- **Operational** : online FS freshness < 200 ms, scoring latency p99 < 150 ms
- **Reliability** : uptime scoring service >= 99.95%

---

15) Checklist rapide pour démarrage (MVP ML)

- Installer Feature Store offline (S3) + online (Redis)
- CDC → Kafka → streaming job → online FS (velocity features)
- Build initial training dataset (last 12 months)
- Train baseline model + register

- Deploy serving in canary + enable logging & monitoring
- Configure drift detectors & alerts