AuraPass — Dokumen Teknis untuk Developer

Dokumen ini memuat rincian arsitektural teknis, komponen, rekomendasi teknologi, dan kontrak operasional yang dibutuhkan developer untuk membangun **Autonomous Credit Passport Engine**. Tujuan: menjadi panduan langsung untuk tim engineering (backend, infra, smart-contract, data/ML).

---

1. Arsitektur Ringkas (Recall)

Sistem terdiri dari pipeline event-driven yang menangkap aktivitas on-chain → normalizer/feature store → reputation engine (heuristic + ML) → ZK proof layer → API & on-chain oracle adapter. Semua komponen modular dan dapat diskalakan horizontal.

---

2. Komponen & Rekomendasi Teknologi

2.1 Indexer / Event Listener

Tugas: capture tx/logs, contract interactions, token transfers, internal tx

Rekomendasi: - **Primary:** The Graph (subgraph) untuk indexing event standar. - **Supplementary / Real-time:** Custom indexer (Node.js with ethers.js or TypeScript) yang berjalan via WebSocket RPC untuk low-latency event capture. - **High-throughput option:** Rust-based indexer (e.g., using `web3` crates) or Substreams if heavy data needs.

Why: The Graph provides easy querying and indexing; custom indexer gives deterministic real-time updates and more control for edge cases.

2.2 Message Queue / Stream Bus

Tugas: buffer event, guarantee delivery, fan-out to ETL and workers

Rekomendasi: - **Primary:** Kafka (Apache Kafka) for guaranteed ordering and high throughput. - **Lightweight alternate:** Redis Streams (if infra simplicity & lower ops cost desired).

Why: Kafka supports retention, partitioning, consumer groups for parallel workers; Redis Streams easier to operate for smaller teams.

2.3 ETL / Normalizer

Tugas: transform raw events into feature vectors (per wallet/time window)

Rekomendasi: - **Language:** Python (pandas) or TypeScript (node) depending on team skillset. - **Runtime:** Dockerized worker services; scale with Kubernetes (K8s) or serverless (e.g. AWS Fargate).

Features to extract (examples): - tx_count_per_day/week - net_token_flow (in/out) - unique_counterparties - leveraged positions (lending protocol interactions) - average_gas_spend - onchain_age - stablecoin_balance_ratio

Output format: standardized JSON schema saved to Feature Store.

2.4 Feature Store & Time-series DB

Tugas: store feature snapshots and historical time-series

Rekomendasi: - **Primary:** ClickHouse (fast analytical reads) + MinIO for raw event blobs. - **Alternative:** TimescaleDB for pure time-series with SQL semantics.

Why: ClickHouse scales for large analytical workloads (millions of wallets). TimescaleDB is simpler for relational time-series queries.

2.5 Reputation Engine (Scoring)

Tugas: deterministic heuristics + ML models produce credit score, tier, and explanation tags

Stack Recommendation: - **Heuristics & Rules:** implement in Python for readability (FastAPI microservice). Keep as fallback if ML service unavailable. - **ML Model Training:** Python (scikit-learn, XGBoost for gradient boosting) for tabular data. Use PyTorch/LightGBM as needed. - **Model Serving:** BentoML or TorchServe (or FastAPI + Uvicorn for simple models). Use MLflow to version models.

Model pipeline: - offline training on historical features - backtesting + cross-validation - store model artifact in model registry (MLflow) - expose predict endpoint for realtime scoring

Explainability: SHAP values for per-wallet feature contributions (cached for API consumption).

2.6 ZK Proof Layer

Tugas: produce ZK proofs for claims (e.g., score $\geq$ X) and create commitments/hashes for on-chain verification

Options & Recommendations: - **Starter (fast):** Circom + snarkjs for threshold circuits (big ecosystem, many examples). - **Alternative (modern):** Noir (for Rust-like developer ergonomics) or Halo2/Plonk for custom circuits.

Design patterns: - keep circuit small (prove threshold only, not raw features) - produce score commitment/hash off-chain and publish nullifier to prevent replay - provide an endpoint `POST /proof/generate` that returns proof and public inputs

Storage: store proofs + metadata in object store (MinIO/S3) with provenance (block height + tx hash).

2.7 API Gateway & Admin Dashboard

Tugas: expose public API to integrators and internal admin UI

API Design (suggested endpoints): - `GET /api/credit/{wallet}` → {score, tier, last_updated, tags} - `GET /api/history/{wallet}` → time-series list of scores - `GET /api/proof/{wallet}?threshold=600` → returns latest proof/meta if proof exists - `POST /api/webhook/indexed` → webhook for indexer notifications (optional)

Stack: - **Gateway:** Nginx / Kong / AWS API Gateway - **Backend:** FastAPI (Python) or Node.js (TypeScript + NestJS) - **Auth:** API Key / OAuth2 for partners; rate-limiting per key

Admin Dashboard: React (TypeScript) with charts (recharts or recharts-like) and role-based access.

2.8 On-chain Oracle Adapter & Smart Contracts

Tugas: provide on-chain interface to consume proofs / or to push score hashes

Smart Contract Patterns: - `ScoreOracle.sol` that stores `wallet -> score_hash` with event emission - `Verifier.sol` for on-chain proof verification (minimized gas cost by verifying only small circuits or public inputs) - Consider *push vs pull* pattern: prefer **pull** (dApp queries off-chain API) initially, and allow **push** for critical workflows.

Stack: Solidity (>=0.8.x), Hardhat for testing & deployment.

Security: unit tests, fuzzing with foundry, and formal verification for critical functions.

2.9 Storage, Backup & Audit

Recommendations: - Raw events & proofs → object store (MinIO / S3) with lifecycle policies. - DB backups → automated snapshots (daily incremental). - Immutable audit ledger → store proof commitments on-chain (small footprint) or use timestamped signed logs.

2.10 Monitoring, Observability & Ops

Recommendations: - **Metrics:** Prometheus + Grafana (system + business metrics: scored wallets, latency, model drift) - **Tracing:** OpenTelemetry + Jaeger - **Logs:** Centralized ELK / Loki stack - **Alerting:** PagerDuty / Opsgenie integration for critical failures

---

2.11 Security & Privacy

Practical Measures: - encryption-at-rest for DB & S3 - TLS everywhere - role-based access for dashboard and key management - rotate API keys; HSM for critical private keys - privacy: store aggregated features and hashed commitments rather than raw PII

Anti-manipulation: - anomaly detection service for suspicious pattern - sybil mitigation: link PoH / SBT badges as attestations - challenge/appeal flow where user can request re-evaluation and provide off-chain proofs

---

3. Data Schemas (Examples)

Feature Vector (sample JSON)

```
{
  "wallet": "0x...",
  "window_start": "2025-11-01T00:00:00Z",
  "tx_count": 45,
  "unique_counterparties": 21,
  "net_flow_usd": 1234.56,
  "stablecoin_ratio": 0.62,
  "avg_gas": 0.0021,
  "onchain_age_days": 450
}
```

Score Document

```
{
  "wallet":"0x...",
  "score": 742,
  "tier":"A",
  "tags":["active_borrower","stable_saver"],
  "confidence":0.87,
  "last_updated":"2025-11-19T12:34:56Z",
  "score_hash":"0xabc...",
  "proof_id":"proof-2025-11-19-..."
}
```

---

4. Development Workflow & CI/CD

Repos: split into micro-repos or mono-repo: `indexer`, `etl`, `reputation-engine`, `api`, `zk-circuits`, `smart-contracts`, `dashboard`.

CI/CD: GitHub Actions or GitLab CI: - run unit tests, linters, contract tests, and build containers - deploy to staging cluster (k8s) with canary - automated DB migrations (Flyway / Alembic)

Infrastructure as Code: Terraform for cloud infra (VPC, K8s, RDS, S3), Helm charts for k8s workloads.

---

5. Testing & Validation

- **Contract Tests:** Hardhat + Foundry fuzz tests
- **Integration Tests:** end-to-end (indexer -> ETL -> scoring -> API)
- **Backtesting:** run scoring engine on historical on-chain snapshots
- **Adversarial Tests:** simulate manipulation attempts, gas spike, bulk wallet creation

---

6. Roadmap Prioritas Teknis (first 3 months)

1. Harden indexer + queue + ETL (reliability).
2. Implement simple heuristic scorer + API (v1).
3. Create basic Circom circuit for threshold proof.
4. Integrate ML pipeline with offline training & basic model serving.
5. Pilot integration with a single DeFi partner (testnet).

---

7. Ops Cost Considerations (high level)

- Indexer + Kafka + ClickHouse are the main recurring cost drivers.
- For early stage, use managed services (Confluent Cloud, Aiven ClickHouse) to reduce ops burden.
- Proof generation CPU usage depends on circuit complexity; keep circuits minimal to reduce cost.

---

8. Security & Audit Plan

- Smart contract audit before mainnet (3rd-party).
- ML model review for bias & edge cases.
- Penetration test for API & infra.

---

9. Appendix: Suggested Tech Stack Summary

- **Indexer:** The Graph + custom Node.js (ethers.js)
- **Queue:** Kafka (or Redis Streams)
- **ETL:** Python workers (pandas)
- **Feature Store:** ClickHouse / TimescaleDB
- **Modeling:** Python (scikit-learn, XGBoost), MLflow
- **Serving:** FastAPI + Uvicorn / BentoML
- **ZK:** Circom + snarkjs (starter), Noir (alternative)
- **Smart Contracts:** Solidity + Hardhat + Foundry
- **Storage:** MinIO / S3
- **Infra:** Docker, Kubernetes, Terraform
- **CI/CD:** GitHub Actions
- **Monitoring:** Prometheus/Grafana, Jaeger, ELK

Jika kamu mau, aku bisa lanjut dengan: 1. **Checklist tugas developer** terperinci (Jira-ready).
2. **Template repo** (readme + skeleton) untuk indexer / API / contracts.
3. **Diagram topologi jaringan** (PNG/SVG).

Pilih salah satu — aku buatkan segera.