🏷️ Generating Wallet Tags for Solana Wallets

Helping investors make smarter decisions through on-chain labels

💡 Problem Statement

> Create a pipeline that produces valuable labels for blockchain addresses. These labels should be useful to an investor who wants to better understand interesting addresses on chain, and help them make smarter decisions.

🎯 User-Centric Design

- **What makes a label valuable?**
  - Helps investors identify high-signal wallets based on PnL and activity patterns.
  - Distinguishes wallet types (e.g., DEX traders, funds, whales, long-term holders).
  - Enables actionable insights by reflecting wallet activity over different time horizons (1h, 1d, 3d, 7d, 30d).
  - Supports segmentation — group wallets with similar behavior for deeper analysis.
  - Facilitates filtering and sorting in UI tools to help users focus on relevant wallets.

- **End-user focus**
    - Tags should be **clear, consistent, and interpretable** — ideally unique and mutually exclusive where possible.
    - Provide a **wide range of tags** tailored to different user needs:
        - *Investors* → `Smart Money`, `Whale`, `Fund`
        - *Protocol builders* → `Heavy User`, `Bot`
        - *Researchers* → `Bridge User`, `Mixing Activity`
    - Insights must be **timely** — labels should reflect *current* wallet activity, updated regularly (1h, 1d, 3d, 7d, 30d)
    - Enable users to **create segments of wallets** by similar patterns for discovery and monitoring.

⚠️ Considerations & challenges

- Finding good quality indexed Solana dataset (not an EVM chain)
- Produce a data model that captures the complex relationship between transactions, accounts, token transfers & instructions
- Ensure the data platform supports high TPS (Solana = high-throughput chain) and high read
- Handle blockchain reorganizations (e.g. forks, chain reorgs) gracefully.
- What are useful tags based on trading behavior, what should the time horizons be? (1h, 1d, 1w, 1m?)
- Prevent feature leakage in tagging models (e.g. avoid double counting signals like transaction count and volume).
- Keep tags fresh without excessive recomputation → balance between batch and incremental updates.

🚀 Milestones & Tasks

Milestones:

1️⃣ Set up streaming + storage
2️⃣ Build aggregated wallet activity models
3️⃣ Wallet heuristics train + deploy wallet tagger
4️⃣ Serve labels to end-users

💎 Tasks per Milestone

- 1️⃣ **Top priority: Streaming + Storage**
  - Connect Goldsky to ClickHouse (append-only mode).
  - Set up base tables with dbt (transactions, instructions, token accounts, tokens, token transfers).
  - Setup orchestration (nice to have)
- 2️⃣ **Aggregation & Features**
  - DBT models: wallet-level aggregates (balance, tx count, token types).
  - Handle time windows (1h, 1d, 3d, 7d, 30d).
- 3️⃣ **Machine Learning**
  - Feature engineering: volume, DEX interactions, clustering.
  - XGBoost classifier + graph embeddings.
- 4️⃣ **Label Serving**
  - API
  - Materialized views for fast queries. Experiment with clustering and partitioning to improve performance on read.

🕰️ Prioritization

- Streaming + storage solution should be top priority, as well as building a sound data model around Solana data to power analytics.
- Building a base transactions table allows us to build a wallet level table with aggregates that help for potential ML feature engineering or a heuristical approach in determining tags
- Chain choice: settle for Solana, one of the chains with the biggest TVL and highest DEX trading activity
- Optional: setup data orchestration of dbt models ➡️ highest priority should lay with building a basic transaction table for iterative testing of assumptions.

🏗️ Architecture

Architecture Diagram

🏗️ Stack

- Clickhouse is a columnar based storage that supports fast reads and fast writes, ideal for blockchain analytical workloads
- Goldsky is a blockchain indexer which can stream or mirror data into your database. Goldsky supports Solana and handles reorgs for you. Caveat: data quality is not the best.
- dbt is a versatile tool for building data models for analytics use cases with version control and good data testing coverage
- Mage.ai is a relatively new data orchestration tool that has native Clickhouse and dbt integration and simplifies a lot of data engineering workload

🏗️ Approach

- Stream indexed Solana data (tokens, accounts, token_transfers, transactions, instructions) into Clickhouse using Goldsky.
- Create a clean transfers table serving as a basis for wallet level aggregates, which will in turn be used for heuristics + feature engineering for supervised/unsupervised ML loads.
- Nice to have: implement orchestration with Mage.ai, but to develop a POC for wallet tagging, it is not crucial, as long as there is data for several days to develop heuristics.
- Transfers table will eventually be rolled up into accounts table, the accounts table holds the initial balances of received tokens ➡️ balance at time (t) = starting balance (t-1) + inflow - outflow. Inflow and outflows should come from transfers table.
- This allows us to calculate PnL over time per wallet and token combination, using FIFO heuristic and a rolling average for the cost price to determine the PnL.

🤑 Smart Money tags

- A final wallets table will hold the following aggregates:
  - Transactions count (column per 1h, 1d, 3d, 7d, 30d)
  - Transaction usd volume (column per 1h, 1d, 3d, 7d, 30d)
  - Nested field with dictionary of tokens and token usd balance
  - One aggregated total usd wallet balance field + ideally single column for BTC, ETH and SOL balance for performant querying.
  - PnL (column per 1h, 1d, 3d, 7d, 30d)
- This allows us to create the following tags:
  - Smart Money: top 10 of consistently profitable wallets across all time horizons (1h, 1d, 3d, 7d, 30d)
  - Smart 30d Dex Trader: top 30% of 30d profitable traders
  - 7D Smart Dex Trader: top 30% of 7d profitable traders
  - Token Millionaire: Has a token balance of at least $1,000,000
  - Active Token Millionaire: Top 1000 Token Millionaires that are most active for the past 30 days (based on 30d transaction count and aggregated token balance)

💀 Challenges faced ➡️ Assumptions made

Goldsky's Solana dataset is not the cleanest, I have encountered the following issues, leading to longer development times:

- Having to rely on Solana instructions (a huge table) as a safe bet, unfortunately the existing token_transfers table does not have an index field like instructions has, indicating the order of solana instructions within a transaction. This is challenging because there is otherwise no way to find out which address initiated a transaction, which authority signed, which address received tokens and in which mint etc, they appear in random order within a transaction.
- Having to join back onto transactions table to check if transactions failed or not based on status field.
- Dealing with edge cases, applying heuristics to create a deterministic data transformation logic to arrive to a consolidated transfers table. The heuristics can be found in the int_transfers table. Assumptions per transaction: first authority encountered from top to bottom indicates wallet sending and receiving tokens, last mint in last index indicates the mint of token received, the destination token account is in the destination field where index = max(index) etc.

🤖 Scalability

- Incremental load vs. batch load. Given the high rate of inserts, high computation costs for wallet aggregation and low latency read requirements for the frontend, I made the decision to incrementally load data, inserting and updating only the latest data, see below.

```
with tr as (
  select
    *,
    date(block_timestamp) as block_date
  from
    {{ ref('stg_instructions') }}
  where
    program = 'spl-token'
    and instruction_type not in ('burn', 'mintto', 'closeaccount')
    {% if is_incremental() %}
    and block_timestamp >= (select max(block_timestamp) from {{ this }})
    {% endif %}
)
```

🤖 Scalability

- To support performant queries, it would be wise to materialize the final wallets table as an **incremental materialized view** in Clickhouse, since this view will contain complex aggregations and therefore it is necessary to avoid frequent recomputation and allow for regular fast queries.

```
{{config(
    materialized='materialized_view',
    engine='MergeTree()',
    order_by='(id)',
    catchup=True
)}}
)
```

🤖 Scalability

- Since Goldsky streams data into Clickhouse in append-only mode using the Clickhouse **ReplaceMergeTree** config, this means updates and deletes are inserted like new records. Deletes carry an is_deleted = 1 flag, weheras updated records are insterted as is. Updates are dealt with asynchronously by the engine. Deletes will have to be dealt with manually. Both can actually be dealt with manually using custom logic, the question is rather if this is necessary given the trade-off of added value vs. complexity.

- Clickhouse: ***While results may be slightly inaccurate for a period if duplicate events are inserted, given the large number of rows and the tiny percentage of duplicates, we expect this to be rarely an issue, with most queries not requiring row-level accuracy.***

- Deletes can be resolved using the following logic in downstream tables:

```
where is_deleted = 0
```

🤖 Scalability

ReplaceMergTree Diagram

🎬 Final outcome

- The int_transfers table is an intermediate table whereby 1 row : 1 token transfer and includes columns for token_out and token_in aggregated across instructions, including the volumes of the trades and token metadata.
- This table will eventually be rolled up with accounts table to be able to calculate token balance over time and Pnl, both are necessary to calculate the tags introduced earlier.