Data Engineering Interview - Take-home Assignment

Theme: On-chain trading bot tracking

Context: The Solana account `MEViEnscUm6tsQRoGd9h6nLQaQspKj7DB2M5FwM3Xvz` holds an on-chain trading program. This program ("contract" in other chains' terms) contains the necessary logic to check for arbitrage opportunities between different pools / swap venues, and execute a trade when one is found.

By design, this program will only execute profitable trades, which means its performance can be evaluated by comparing its "burn" (i.e., the amount of SOL it spends sending transactions, regardless of success or failure) to the amount of SOL it earns from its trades.

You are a trader yourself and discovered this competitor bot while digging through the explorer. You need to gain insight into its operations.

Breakdown

Part 1 - Ingest

Objective: Write an indexing program in Rust which:

- subscribes and listens to the bot's raw on-chain data stream from an RPC provider
- tracks the bot's burn
- stores that burn in a relational database for consumption

Resources:

- [RPC streaming service doc](https://docs.triton.one/project-yellowstone/dragons-mouth-grpc-subscriptions) - Docs on the data streaming provider, which contains consumer examples that make for good starting points.
- RPC credentials:
  - JSON-RPC URL: "https://temporal.rpcpool.com/4586b536-c930-43f1-91c1-bee31ab3a0a2"
  - gRPC credentials:
    - endpoint: "https://temporal.rpcpool.com"
    - x-token: "4586b536-c930-43f1-91c1-bee31ab3a0a2"
- [NeonDB](https://neon.com/) offers a free tier that will allow you to create a database and ingest enough records for the purpose of this assignment
- [Solscan](https://solscan.io/) is a Solana explorer that displays valuable information about blocks, transactions, accounts, etc.
- [Solana JSON-RPC docs](https://solana.com/docs/rpc).

Notes:

- If you encounter any technical issues with Neon's free tier, feel free to fall back on a local relational database of your choice.

Part 2 - Visualize

Objective: Build a dashboard displaying relevant insights into the bot's burn / operations.

Resources:

- [Grafana Cloud](https://grafana.com/): offers a free trial that allows you to create a dashboard and share a link with anyone.

Notes:

- In case there was any technical issue with Neon's free tier in the previous part, or with Grafana Cloud in this part, feel free to fallback on creating charts/plots using matplotib or any equivalent tool that will allow sharing results.

Part 3 - Expand

Objective: Ingest additional columns / tables for a couple other datapoints you deem relevant, and create the corresponding displays.

Deliverables

- A GitHub repo containing the final code for the indexing program
- Credentials (host/password) to the test database if publicly available, or a sample CSV dump of relevant tables if issues were encountered with NeonDB
- A link to the dashboard if publicly available, or screenshots of the charts if issues were encountered with Grafana Cloud

Final notes

- The instructions above are fairly open-ended, feel free to make design choices where requirements do not specify any.
- Although trade revenue is a critical part of the tracking described, it is not explicitly required due to completion time considerations. If you are able to include it regardless, you can do so as part of **Part 3**.
- Feel free to write down your thoughts on anything relevant inside the repo (comments, separate mkdwn file), e.g.:
  - Notes about the implementation / shortcuts taken deliberately

  - Potential blockers
  - Design choices
  - Implementation plan on anything you lacked time to get to and deem important
- Anything giving insight into your thought process is welcome, so don't hesitate to commit to the repo any intermediate steps you used, e.g. unit tests on a single tx to figure out the data structure, etc.