

Monitoring Solana Wallet (Part 1)

Dalam ekosistem kripto, memantau aktivitas wallet tertentu, seperti wallet whale atau untuk copy trading, adalah hal yang umum dilakukan. Dengan menggunakan RPC dari QuickNode, kita bisa mengakses data transaksi detail seperti perubahan saldo sebuah wallet dan **bikin sistem wallet monitoring dan copy trading bot sendiri**. Di sini kita akan fokus belajar untuk chain Solana, jadi sebelum masuk ke code, akan dijelaskan beberapa konsep penting.

Contents

Prerequisites	3
Concept.....	3
System Program	3
Solana's Account Model and Token Accounts.....	3
Mint Account.....	3
Associated Token Account (ATA)	3
Raydium Liquidity Pools (LP)	3
Coding Guide.....	4

Prerequisites

- Pemahaman tentang smart contract dan blockchain dasar
 - Pemahaman tentang Node JS
 - Pemahaman cara kerja Liquidity Pool di Solana

Concept

Sifat blockchain Solana berbeda dengan EVM. Salah satu yang menonjol adalah Solana perlu ada rent deposits untuk setiap account di chainnya agar bisa digunakan. Maka biasanya, kita ngga bisa narik seluruh Solana dari account kecuali di burn. Jumlahnya kecil, paling 0.002 SOL per akun, tapi kalau punya banyak cukup buat Grab Food. 😊

System Program

System Program di Solana adalah program inti yang digunakan untuk basic operation seperti transfer SOL, account creation, dan mengurus rent fee. Alamat dari System Program di Solana adalah:

Banyak transaksi di Solana berinteraksi dengan alamat ini, tetapi biasanya tidak relevan saat kita memantau token spesifik seperti SPL tokens dalam Liquidity Pool (LP).

Solana's Account Model and Token Accounts

Di Solana, **wallet** adalah account milik **System Program** dan sistem kepemilikan tokennya berbeda dari blockchain berbasis EVM seperti Ethereum. Di EVM-based chain, semua token terhubung ke 1 account address, dan saldo token dicek menggunakan fungsi **balanceOf** dari token contract. Sementara itu, Solana menggunakan **subaccount** terpisah yang disebut **token accounts** untuk setiap jenis token yang dimiliki oleh sebuah main account atau **pubkey**. Jadi, jika sebuah wallet di Solana memiliki beberapa jenis token, akan ada **subaccount terpisah** untuk masing-masing token tersebut. Konsep ini memungkinkan pengelolaan saldo yang lebih fleksibel dan terpisah.

Mint Account

Mint adalah istilah yang digunakan di Solana untuk merujuk pada token jenis tertentu atau contract yang bertanggung jawab untuk mengelola creation dan distribution token tersebut. Setiap token di Solana memiliki Mint yang unik, yang mengelola atribut seperti **token supply, decimals, dan account roles (mint-authority dan freeze-authority)**.

Associated Token Account (ATA)

Associated Token Account (ATA) adalah sebuah account khusus di Solana yang digunakan untuk menyimpan token tertentu seperti **token accounts**. Cara kerjanya sedikit berbeda dimana dengan token accounts, penerima perlu punya token account untuk bisa menerima sebuah token, sedangkan di ATA, hal ini tidak diperlukan karena address ATA ditentukan otomatis dari mint dan owner account sehingga 1 owner hanya ada 1 ATA khusus 1 token. Beda dari token account yang bisa saja ada lebih dari 1 untuk 1 token.

Raydium Liquidity Pools (LP)

Solana tidak ada single standard LP seperti EVM yang mostly fork dari Uniswap. Jenis LP di Solana ada banyak tapi kita akan meng-cover 2 yang paling umum dari Raydium.

- a. Concentrated Liquidity Market Maker(CLMM)

LP contract punya 2 token account untuk pair-nya. Umumnya WSOL (wrapped SOL) dan untuk mint account. Owner contract ini adalah:

CAMMCzo5YL8w4VFF8KVHrK22GGUsp5VTaW7grrKgrWqK

Jadi, jika ada transaksi yang melibatkan contract yang owner-nya address di atas, maka itu adalah sebuah LP. Biasa nama contract LP-nya menggunakan format “Raydium (PAIR) Market”

Overview		More info	
SOL Balance	0.011637 SOL (\$1.7641)	Public name	Raydium (SOL-BILLY) Market
Token Balance		Owner	Raydium Concentrated Liquidity 🔗
 WSOL	540.03 (\$82,171.68)	isOnCurve ⓘ	False
 BILLY	9,863,133.49 (\$524,235.4)	Tags	raydium 2+

- b. ATA owned by Raydium Authority (currently V4 or version 4)

Jadi address Raydium Authority ini (5Q544fKrFoe6tsEbD7S8EmxGTJYAKtTVhAW5Q5pge4j1) memiliki banyak ATA untuk setiap token LP. Kalau di lihat portfolio-nya, namanya mengandung kata “Pool”.

Account	Token	Token Balance
Raydium (MAGAPEPE-SOL) Pool 1 🔗	 MAGAPEPE	3,550,900,624,925,346
Raydium (SOLPAC-SOL) Pool 1 🔗	 SOLPAC	6,253,784,891,048,510
Raydium (CHWY-SOL) Pool 1 🔗	 CHWY	4,761,334,899,754,892

Karena LP menyediakan likuiditas, cara kerjanya, jika token dibeli dari LP dengan SOL, maka SOL di LP bertambah dan token di LP berkurang, berlaku sebaliknya.

Feel free to try it yourself before reading the guide.

Coding Guide

Di sini kita pakai approach API job dengan Cron sampai bisa mendapatkan transaksi yang dilakukan wallet.

1. Import Libraries

Pertama-tama, kita butuh beberapa library penting seperti `@solana/web3.js` untuk berinteraksi dengan blockchain Solana, `@project-serum/anchor` untuk keperluan wallet management, `node-cron` buat scheduling task otomatis, dan `bs58` untuk encoding. Kita juga perlu metadata dari program `@metaplex/js` yang membantu untuk mengakses informasi token.

```
import * as solanaWeb3 from '@solana/web3.js';
import { Wallet } from '@project-serum/anchor';
import cron from 'node-cron';
import bs58 from 'bs58';
import { programs } from '@metaplex/js';
const { Metadata } = programs.metadata;
```

2. Setup Account Addresses

Di sini, declare variable wallet yang dimonitor, address Wrapped SOL dan Raydium LP owners agar bisa tau kalau transaksinya berhubungan dengan LP (ada swap).

```
const walletToMonitor = new solanaWeb3.PublicKey('address-yang-dimonitor');
const WRAPPED_SOL_MINT = 'So11111111111111111111111111111111111111111111111111111111111112';
const RAYDIUM_LP OWNERS = [ 'CAMMCzo5YL8w4VFF8KVHrK22GGUsp5VTaW7grrKgrWqk',
'5Q544fKrFoe6tsEbD7S8EmxGTJYAKtTVhAW5Q5pge4j1' ];
```

3. Connect to Solana

Untuk bisa memantau aktivitas di blockchain, kita butuh koneksi ke node Solana. Disini kita pakai RPC dari [QuickNode](#) yang bisa kalian bikin secara gratis, perlu hubungkan kartu kredit saja dan pilih free plan. **Best practice, data yang pribadi seperti private key dan private RPC ditaruh di file .env.**

```
const connection = new solanaWeb3.Connection("URL-RPC-QUICKNODE", { commitment: 'confirmed',
confirmTransactionInitialTimeout: 60000 });
```

commitment adalah parameter yang menentukan tingkat finalisasi sebuah transaksi atau state di jaringan Solana yang kita inginkan ketika kita melakukan permintaan data. Di sini kita set ke confirmed untuk memastikan transaksi yang diambil sudah dikonfirmasi oleh mayoritas validator (66%+). confirmTransactionInitialTimeout adalah waktu maksimal (dalam millisecond) untuk menunggu konfirmasi pertama dari transaksi setelah dikirim. Ini berguna kalau nanti kita mau bertransaksi (misal copy trade).

4. Fetch Transactions and Initial Data Processing

Langkah pertama adalah mengambil daftar transaksi yang terkait dengan wallet yang sedang dipantau. Di sini, kita hanya berfokus pada pengambilan dan pengolahan awal dari transaksi tanpa menyelam terlalu dalam ke detail transaksinya. Cek comment di code buat penjelasan singkat per code section.

```
async function monitorWallet() {
  try {
    // Ambil daftar transaksi terkait dengan wallet yang dipantau
    let txnList = await connection.getSignaturesForAddress(walletToMonitor);

    if (txnList.length > 0) {
      // Ambil daftar signature dari transaksi tersebut
      let sigList = txnList.map(txn => txn.signature);

      // Ambil detail transaksi yang terkait dengan signature
      let txnDetails = await connection.getParsedTransactions(sigList, {
        maxSupportedTransactionVersion: 0 });

      console.log("Transaction fetched!");

      // Proses transaksi satu per satu
      for (let i = 0; i < txnDetails.length; i++) {
        const txn = txnList[i];
        const meta = txnDetails[i].meta;

        // Cek apakah ada perubahan saldo token sebelum atau sesudah transaksi
        if (meta.preTokenBalances.length > 0 || meta.postTokenBalances.length > 0) {
          // Olah penulisan tanggal (preferensi aja)
          const date = new Date(txn.blockTime * 1000);
          const formattedDate = new Intl.DateTimeFormat('en-US', {
            day: '2-digit',
            month: '2-digit',
            year: '2-digit',
            hour: '2-digit',
            minute: '2-digit',
            second: '2-digit'
          }).format(date);
          console.log(`Transaksi ${i + 1}: ${formattedDate}`);
        }
      }
    }
  } catch (err) {
    console.error(err);
  }
}
```

```

        month: 'short',
        year: 'numeric',
        hour: '2-digit',
        minute: '2-digit',
        second: '2-digit',
        hour12: false,
    }).format(date);

    console.log(`Txn #${i + 1} by ${walletToMonitor.toString()}`);
    console.log(formattedDate);

    // Ambil accountKeys (accounts yang terlibat di transaksi)
    const accountKeys = txnDetails[i].transaction.message.accountKeys;
    console.log(accountKeys);

    // Cek perubahan balance dan owner setiap account
    for (let j = 0; j < accountKeys.length; j++) {
        const accountKey = accountKeys[j].pubkey;

        if (accountKey !== solanaWeb3.SystemProgram.programId) {
            const preBalance = meta.preTokenBalances.find(a => a.accountIndex
                === j);
            const postBalance = meta.postTokenBalances.find(a => a.accountIndex
                === j);

            if (preBalance || postBalance) {
                const balance = postBalance ? postBalance : preBalance;
                const owner = balance.owner;

                // Lanjut bikin function untuk cek metadata token
            }
        }
    }

    console.log("=".repeat(50));
}

} catch (err) {
    console.log("Error: ", err.message);
}
}

```

- **getSignaturesForAddress:** Mengambil daftar signature transaksi dari sebuah alamat, yang digunakan untuk melacak semua transaksi yang melibatkan alamat tersebut.
- **getParsedTransactions:** Mengambil detail lengkap dari transaksi berdasarkan signature yang diberikan, termasuk informasi akun dan perubahan saldo yang diparsing menjadi format yang mudah dibaca.

5. Get Token Metadata

```

async function getTokenMetadata(mint) {
    try {
        const metadataPDA = await Metadata.getPDA(mint);
        const metadata = await Metadata.load(connection, metadataPDA);
        return metadata.data.data;
    } catch (error) {
        console.error(`Error fetching metadata for mint ${mint}: `, error);
        return 'Unknown Token';
    }
}

```

```
}
```

- **Metadata.getPDA**: Mendapatkan Program Derived Address (PDA), yaitu address unik untuk mengakses metadata dari token tertentu di Solana. Parameternya PublicKey dari mint.
- **Metadata.load**: Memuat metadata token berdasarkan PDA yang telah diperoleh, memungkinkan akses ke informasi penting seperti name, symbol, dan decimal. Panggil di dalam fungsi sebelumnya.

```
if (preBalance || postBalance) {  
    const balance = postBalance ? postBalance : preBalance;  
    const owner = balance.owner;  
    const tokenMetadata = await getTokenMetadata(new  
solanaWeb3.PublicKey(balance.mint));  
  
    console.log(tokenMetadata);  
  
    // Proses lebih lanjut, seperti logging atau keputusan berdasarkan metadata  
}
```

6. Setup CronJob

Kita mau monitoring wallet ini berjalan terus-menerus tanpa harus kita running manual setiap saat. Jadi, kita pakai CronJob yang bakal menjalankan fungsi `monitorWallet` setiap 10 detik. Interval bisa diatur sesuai preferensi dan jangan lupa panggil function `start`. Untuk testing, bisa langsung panggil `monitorWallet` saja.

```
async function start() {  
    try {  
        await cron.schedule("*/10 * * * *", async () => {  
            await monitorWallet();  
        });  
    } catch (err) {  
        console.log("Error: ", err.message);  
    }  
}  
  
start();
```

Selesai sampai tahap membedah transaksi tahap awal.