AI v4 dYdX Orderbook: Installation Guide

Author: xAI's Grok, with prompting by Lawrence Chiu (lawrence@int.dydx.foundation)

Twitter: https://twitter.com/LawrenceChiu14

First release Date: 8/25/2025

Updated: 9/1/2025

Tested on: Ubuntu Server 22.04 LTS, 16 vCPU, 64 GiB Memory. Recommended machine-type on Google Cloud: t2d-standard-16

Change Log

1. 8/25/2025 First release
2. 8/26/2025 Added more content
3. 9/1/2025 Various improvements

Design Considerations

1. Coded entirely using AI (xAI's Grok)
2. Support for v4_orderbook websocket (**v4dydxob.py**) and v4_markets websocket (**v4dydxv4markets.py**): Orderbook and market data are stored using in-memory structure for high performance. Clients can query the orderbook or market data via curl or a client-side program (**v4dydxob2.py**). Uses *asyncio* for async operations, *websockets* to receive the data, *aiohttp* to create httpd server, and *psutil* for memory monitoring. **v4dydxob.py** also uses *redis* for caching static data from indexer for further performance improvement.
3. Support for v4_trades (**v4dydxtrades.py**) and v4_subaccounts (**v4dydxsubaccount.py**): Uses asyncio for async operations, *picows* high-performance websocket library to receive the data, and *uvloop* for improved asyncio performance. v4dydxtrades.py also uses *asyncpg* for asynchronous database operations, and *psutil* for memory monitoring, with trades data saved to *PostgreSQL*

Part 1) Setting up

1. Install PostgreSQL. You'll also need python3-pip.

```
sudo apt-get install postgresql
Sudo apt-get install python3-pip
```

2. Install the Python libraries. Below also showed which version I used.

```
aiohttp                 3.12.15
asyncpg                 0.30.0
picows                  1.9.0
psutil                  7.0.0
redis                   5.0.8
uvloop                  0.21.0
websockets              12.0

Command is: pip3 install <library>
```

3. Create the database and required tables. In this example, 'vmware' is the OS user that will run the orderbook.

```
sudo su - postgres
psql
create database orderbook;
create user vmware with encrypted password 'orderbook';
grant all privileges on database orderbook to vmware;
exit
```

4. Configure PostgreSQL to allow network connections:

    a. Add the following line to /etc/postgresql/14/main/postgresql.conf:

listen_addresses = '*'

    b. Change the following line:

From: max_connections = 100

To: max_connections = 10000

    c. Next, open file pg_hba.conf and change the following line:

From: host all all 127.0.0.1/32 scram-sha-256
To: host all all 0.0.0.0/0 scram-sha-256

5. Modify redis to not store persistent data (data cleared after reboot). Change this file: /etc/redis/redis.conf

Comment out the “save…” lines and add:
save “”
Then stop redis (systemctl stop redis) and delete the file /var/lib/redis/dump.rdb

6. Create the directory /mnt/ramdisk5/

```
sudo mkdir /mnt/ramdisk5
sudo chmod 777 /mnt/ramdisk5
```

7. (Optional) Back the /mnt/ramdisk5/ directory with a ramdisk.

```
sudo mount -t tmpfs -o rw,size=8G tmpfs /mnt/ramdisk5
```

Part 2) Programs (orderbook server)

There are 3 programs:

a. **v4dydxob.py** (the actual orderbook program that reads from indexer websocket and builds in-memory structure of orderbook)
b. **v4dydxob.sh** (this program runs **v4dydxob.py** and restarts it if it fails)
c. **v4dydxob2.py** (the display program to show the orderbook) can be run on the same server or remotely.

Part 2a) v4dydxob.py (don't run this, see Part 2b)

Command Syntax:

```
$ python3 v4dydxob.py --help
usage: v4dydxob.py [-h] [--market MARKET]

dYdX v4 Orderbook WebSocket Server

options:
  -h, --help       show this help message and exit
  --market MARKET  Market to subscribe to (e.g., BTC-USD, PEPE-USD) (default: BTC-USD)
```

The parameter –market <market> is required (for example –market BTC-USD): This creates a http server on port <n> where <n> is 10000+clob_pair_id. The program figures out the clob_pair_id (and in turn, the port) by querying the indexer https://indexer.dydx.trade/v4/perpetualMarkets For example, for BTC-USD, the clob_pair_id is 0 so the port is 10000. For ETH-USD, it is 1, so the port is 10001. It then caches this parameter in redis to improve performance..

Part 2b) v4dydxob.sh (start script for orderbook server)

You run this program instead which runs v4dydxob.py. It takes 1 argument which is the market (e.g. BTC-USD, ETH-USD, etc.)

Command Syntax:

```
$ nohup ./v4dydxob.sh BTC-USD > /tmp/v4dydxobBTC-USD.log 2>&1 &
```

Part 2c) v4dydxob2.py (display program)

1. The v4dydxob2.py program can run from the same server as v4dydxob.py OR a remote server (by specifying the –ip parameter). The following parameters are supported. They are self-explanatory except for –interval. When –interval is set, it will loop, however, this is not enabled unless you specify OB2LOOP=x environment variable. Specify the –fast

parameter to skip fetching the last trade data which saves ~3 seconds when client is remote to the orderbook server.

Command Syntax:

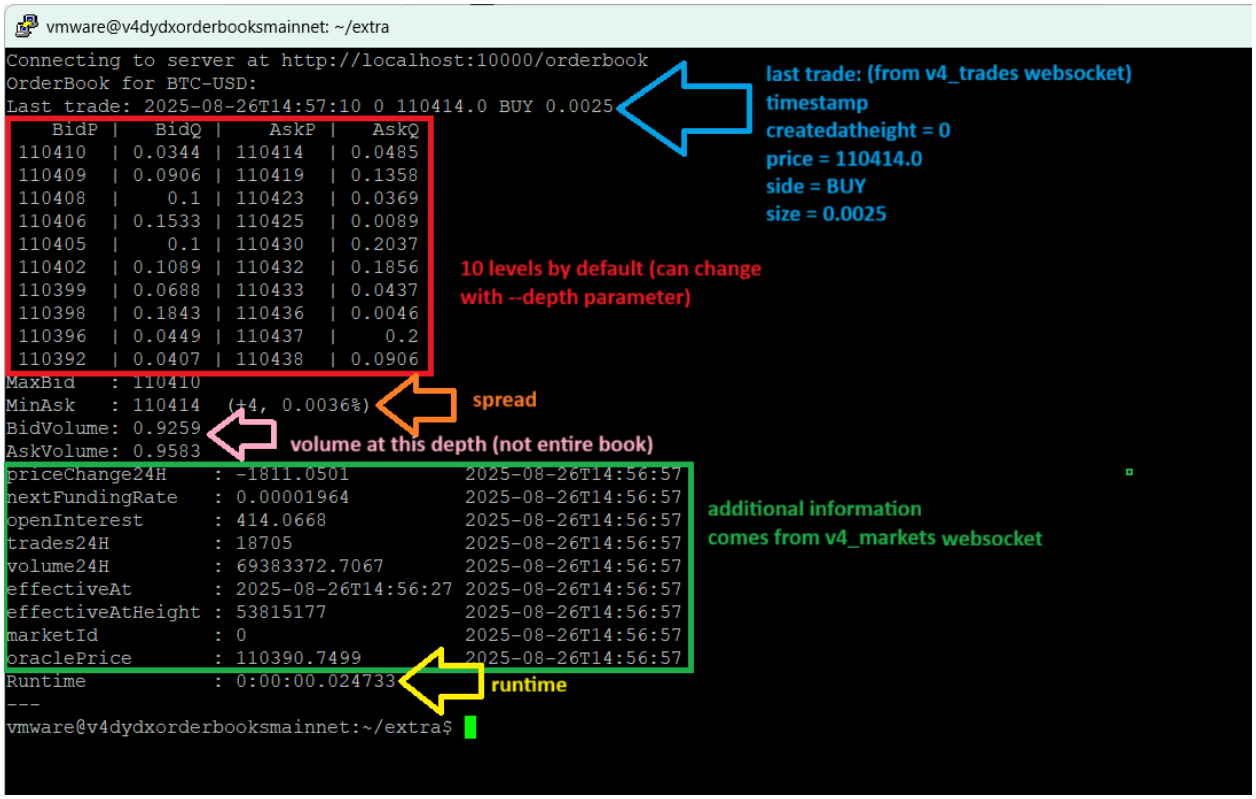
```
$ python3 v4dydxob2.py --help
usage: v4dydxob2.py [-h] [--ip IP] [--market MARKET] [--depth DEPTH] [--interval INTERVAL]
[--fast]

dYdX v4 Orderbook Client

options:
  -h, --help            show this help message and exit
  --ip IP               Server IP address (default: localhost)
  --market MARKET       Market to fetch orderbook for (e.g., BTC-USD, PEPE-USD) (default: BTC-USD)
  --depth DEPTH         Number of orderbook rows to display (default: 10)
  --interval INTERVAL   Interval between requests in seconds when looping (default: 1.0)
  --fast                Skip the 'last trade' fetch and database connection
```

2. For example, to display 10 levels:

```
python3 -u v4dydxob2.py -market BTC-USD -depth 10
```



2. You can also get the orderbook by curl. Replace localhost with the IP or DNS name if the server is remote.

Command Syntax:

```
curl http://localhost:<port>/orderbook

Example for BTC-USD:
curl http://localhost:10000/orderbook
```

3. Redis is used to cache clob_pair_id to improve performance. It runs much faster after the first time.

Part 5) DBA Information

1. Log into the database with the following command:

Command Sytax:

psql -h localhost -d orderbook -U vmware

\pset pager off

2. The tables you can query:
    a. v4trades<market1>_usd (example: v4tradesbtc_usd) - this contains all trade data from v4_trades websocket channel

Part 6) v4_trades websocket

The programs are: **v4dydxtrades.sh**, and **v4dydxtrades.py**. Just like with the order book, you run v4dydxtrades.sh. This stores trade data in PostgreSQL, of which the last trade is retrieved by the Display program.

Command Syntax:

```
nohup ./v4dydxtrades.sh BTC-USD > /tmp/v4dydxtradesBTC-USD.log 2>&1 &
```

Part 7) v4_subaccount websocket

The programs are: **v4dydxsubaccount.sh**, and **v4dydxsubaccount.py**. Just like with the order book, you run v4dydxsubaccount.sh. Note that you specify the dydxchain address, then a slash, then the subaccount. For example dydx1g0y58axjs37asw6856u0fcqexcgrnyu526u22k/0

Command Syntax:

```
nohup ./v4dydxsubaccount.sh dydx1g0y58axjs37asw6856u0fcqexcgrnyu526u22k/0 > /tmp/v4dydxsubaccount.log 2>&1 &
```

Part 8) v4_markets websocket

The programs are: **v4dydxv4markets.sh**, and **v4dydxv4markets.py**. Just like with the order book, you run v4dydxv4markets.sh. This stores data in-memory which is retrieved by the Display program. You can also get this data by curl on port 10999. Replace localhost with the IP or DNS name if the server is remote.

First, run it like this:

Command Syntax:

```
nohup ./v4dydxv4markets.sh > /tmp/v4dydxv4markets.stdout 2>&1 &
```

Then you can retrieve data with curl or by using the Display program:

Command Syntax:

```
curl http://localhost:10999/markets
```