**Indexing using Subsquid**

**Source:** https://docs.sqd.dev/sdk/tutorials/substrate/

**Step 1. Environment Setup**

1.1 Install Node.js version 20 or newer.

npm install n –g

n stable

n latest

node –version

1.2 Install Squid CLI

npm i -g @subsquid/cli@latest

sqd –version

1.3 Install Docker

sudo apt-get update

sudo apt-get install ca-certificates curl

sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \

$(. /etc/os-release && echo "$VERSION\_CODENAME") stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

docker --version

**Step 2. Create Squid Project**

2.1 Use sqd init and come up with some unique name for your squid.

sqd init substrate-xode-tutorial --template substrate

cd substrate-xode-tutorial

2.2 Run the Project

npm i

npm run build

docker compose up –d

If error: run sudo systemctl start docker

npx squid-typeorm-migration apply

node -r dotenv/config lib/main.js # will begin ingesting blocks

# open a separate terminal for this next command

npx squid-graphql-server # should begin listening on localhost:4350/graphql

After this test, shut down both processes with Ctrl-C and proceed.

**Step 3. Define the schema and generate entity classes**

3.1 We make changes to the data schema of the squid and define entities that we would like to track. We are interested in:

* Active Accounts;
* Balance Transfer

For this, we use the following schema.graphql:

Schema.graphql

type Account @entity {

"Account address"

id: ID!

transfersTo: [Transfer!] @derivedFrom(field: "to")

transfersFrom: [Transfer!] @derivedFrom(field: "from")

}

type Transfer @entity {

id: ID!

blockNumber: Int! @index

timestamp: DateTime! @index

extrinsicHash: String @index

from: Account!

to: Account!

amount: BigInt! @index

fee: BigInt! # fee is calculated at the best effort and may be zero for some old extrinsics

}

}

3.2 To finalize this step, run the codegen tool:

npx squid-typeorm-codegen

This will automatically generate TypeScript entity classes for our schema. They can be found in the src/model/generated folder of the project.

**Step 4. Generate TypeScript wrappers for events**

4.1 Obtain metadata of xode.

To obtain medata.json, use this command

npx squid-substrate-metadata-explorer --rpc wss://rpcnodea01.xode.net/n7yoxCmcIrCF6VziCcDmYTwL8R03a/rpc --out metadata.json

4.2 Configure typegen.json. Open the file and follow this:

{

"outDir": "src/types",

"specVersions": "metadata.json",

"pallets": {

"Swork": {

"events": [

"WorksReportSuccess",

"JoinGroupSuccess"

]

},

"Market": {

"events": [

"FileSuccess"

]

}

}

}

Set the "specVersions" field to a valid source of Xode chain runtime metadata.

4.3 Once done with the configuration, we run the tool with

npx squid-substrate-typegen ./typegen.json

The generated Typescript wrappers are at src/types.

**Step 5. Set up the processor object**

5.1 Create a SubstrateBatchProcessor object which subscribes to all the events we need. We do it at src/processor.ts:

Just Set the RPC Endpoint to Xode Endpoint

src/processor.ts

import {assertNotNull} from '@subsquid/util-internal'

import {

BlockHeader,

DataHandlerContext,

SubstrateBatchProcessor,

SubstrateBatchProcessorFields,

Event as \_Event,

Call as \_Call,

Extrinsic as \_Extrinsic

} from '@subsquid/substrate-processor'

import {events} from './types'

export const processor = new SubstrateBatchProcessor()

.setRpcEndpoint({

url: 'wss://rpcnodea01.xode.net/n7yoxCmcIrCF6VziCcDmYTwL8R03a/rpc',

rateLimit: 10

})

.addEvent({

name: [events.balances.transfer.name],

extrinsic: true

})

.setFields({

event: {

args: true

},

extrinsic: {

hash: true,

fee: true

},

block: {

timestamp: true

}

})

// Uncomment to disable RPC ingestion and drastically reduce no of RPC calls

//.useArchiveOnly()

export type Fields = SubstrateBatchProcessorFields<typeof processor>

export type Block = BlockHeader<Fields>

export type Event = \_Event<Fields>

export type Call = \_Call<Fields>

export type Extrinsic = \_Extrinsic<Fields>

export type ProcessorContext<Store> = DataHandlerContext<Store, Fields>

Install required dependencies

npm install @subsquid/util-internal

npm install @subsquid/substrate-processor

5.2 Define the batch handler. Open src/main.ts and follow this:

import {TypeormDatabase, Store} from '@subsquid/typeorm-store'

import {In} from 'typeorm'

import \* as ss58 from '@subsquid/ss58'

import assert from 'assert'

import {processor, ProcessorContext} from './processor'

import {Account, Transfer} from './model'

import {events} from './types'

processor.run(new TypeormDatabase({supportHotBlocks: true}), async (ctx) => {

let transferEvents: TransferEvent[] = getTransferEvents(ctx)

let accounts: Map<string, Account> = await createAccounts(ctx, transferEvents)

let transfers: Transfer[] = createTransfers(transferEvents, accounts)

await ctx.store.upsert([...accounts.values()])

await ctx.store.insert(transfers)

})

interface TransferEvent {

id: string

blockNumber: number

timestamp: Date

extrinsicHash?: string

from: string

to: string

amount: bigint

fee?: bigint

}

function getTransferEvents(ctx: ProcessorContext<Store>): TransferEvent[] {

let transfers: TransferEvent[] = []

const ss58Codec = ss58.codec(42); // Replace 42 with your chain's SS58 prefix.

for (let block of ctx.blocks) {

for (let event of block.events) {

if (event.name == events.balances.transfer.name) {

let rec: {from: string; to: string; amount: bigint; asset: string}

if (events.balances.transfer.v1020.is(event)) {

let [from, to, amount] = events.balances.transfer.v1020.decode(event)

rec = {from, to, amount, asset: "SomeAsset"} // Example: Set asset to a fixed value or decode it if available

} else if (events.balances.transfer.v1050.is(event)) {

let [from, to, amount] = events.balances.transfer.v1050.decode(event)

rec = {from, to, amount, asset: "SomeAsset"}

} else if (events.balances.transfer.v9130.is(event)) {

rec = events.balances.transfer.v9130.decode(event)

rec.asset = "SomeAsset" // Set the asset value here if it's missing

} else {

throw new Error('Unsupported spec')

}

assert(block.header.timestamp, `Got an undefined timestamp at block ${block.header.height}`)

transfers.push({

id: event.id,

blockNumber: block.header.height,

timestamp: new Date(block.header.timestamp),

extrinsicHash: event.extrinsic?.hash,

from: ss58Codec.encode(rec.from), // Custom codec for your chain

to: ss58Codec.encode(rec.to), // Custom codec for your chain

amount: rec.amount,

fee: event.extrinsic?.fee || 0n,

asset: rec.asset // Store the asset as a string directly

})

}

}

}

return transfers

}

async function createAccounts(ctx: ProcessorContext<Store>, transferEvents: TransferEvent[]): Promise<Map<string,Account>> {

const accountIds = new Set<string>()

for (let t of transferEvents) {

accountIds.add(t.from)

accountIds.add(t.to)

}

const accounts = await ctx.store.findBy(Account, {id: In([...accountIds])}).then((accounts) => {

return new Map(accounts.map((a) => [a.id, a]))

})

for (let t of transferEvents) {

updateAccounts(t.from)

updateAccounts(t.to)

}

function updateAccounts(id: string): void {

const acc = accounts.get(id)

if (acc == null) {

accounts.set(id, new Account({id}))

}

}

return accounts

}

function createTransfers(transferEvents: TransferEvent[], accounts: Map<string, Account>): Transfer[] {

let transfers: Transfer[] = []

for (let t of transferEvents) {

let {id, blockNumber, timestamp, extrinsicHash, amount, fee} = t

let from = accounts.get(t.from)

let to = accounts.get(t.to)

transfers.push(new Transfer({

id,

blockNumber,

timestamp,

extrinsicHash,

from,

to,

amount,

fee,

}))

}

return transfers

}

Install required dependencies

npm install @subsquid/typeorm-store

npm install @subsquid/ss58

After this, save the file.

**Step 6. Modify .env file and processor.js**

Change the RPC ENDPOINT to Xode Endpoint

Modify lib/prossesor.js file like this:

Save the files.

**Step 7. Apply changes to the database**

7. 1 Begin by making sure that the database is at blank state:

docker compose down

docker compose up –d

Then we replace any old migrations with the new one with

rm -r db/migrations

npx squid-typeorm-migration generate

The new migration will be generated from the TypeORM entity classes we previously made out of schema.graphql. Apply it with:

npx squid-typeorm-migration apply

7.2 Launce the Project

It's finally time to run the project! Run

node -r dotenv/config lib/main.js

in one terminal, then open another one and run

npx squid-graphql-server

Now you can see the results by visiting localhost:4350/graphql in a browser and accessing the GraphiQL console.

**Result:** Syncing to the Xode Blocks.

GraphQL

To query transfer data, open the "Transfer" section, select the checkboxes for "amount" and "blockNumber," then click the play button next to GraphQL. The queried data will appear on the right side of the screen.

Go to PolkadotJS UI and Select Xode Network. Search for the blocks that appear from the GraphQL. For example, blockNumber 398.

Click the Arrow button beside the search bar.

You can see that in that block there was an event balances.Transfer. So, the indexing successfully connected to the Xode Network.

**Script for running Subsquid**

#!/bin/bash

# Define the command to run

CMD="node –max-old-space-size=16384 -r dotenv/config lib/main.js"

# Specify a log file to capture output

LOG\_FILE="process.log"

# Time in seconds to consider the process as stuck

TIMEOUT=300

# Time in seconds to check for issues

CHECK\_INTERVAL=15

# Function to check if the process is running

is\_running() {

pgrep -f "$CMD" > /dev/null

}

# Function to restart the process

restart\_process() {

echo "Starting process..."

$CMD > "$LOG\_FILE" 2>&1 &

PID=$!

echo "Process started with PID: $PID"

}

# Function to check if the logs are stuck

check\_logs\_stuck() {

last\_mod\_time=$(stat -c %Y "$LOG\_FILE")

sleep "$CHECK\_INTERVAL"

new\_mod\_time=$(stat -c %Y "$LOG\_FILE")

if [ "$last\_mod\_time" -eq "$new\_mod\_time" ]; then

echo "Logs are stuck. Restarting..."

return 1

fi

return 0

}

# Function to check for connection failures in logs

check\_connection\_failure() {

if grep -qE "connection failure|RpcConnectionError" "$LOG\_FILE"; then

echo "Connection failure detected. Restarting..."

return 1

fi

return 0

}

# Start the process initially

restart\_process

# Keep monitoring the process

while true; do

sleep "$CHECK\_INTERVAL"

if ! is\_running; then

echo "Process stopped. Restarting..."

restart\_process

else

check\_logs\_stuck || { kill "$PID"; restart\_process; }

check\_connection\_failure || { kill "$PID"; restart\_process; }

fi

done