Core tools

Node

The node is the core component of the Vector ecosystem. It is responsible for validating transactions, creating blocks, and participating in the consensus protocol. The node is written in Haskell and is open source. You can find the source code here.

In order for you to run this we have multiple options available, you can either run it locally (through docker or a precompiled static binary) or you can use Demeter's CLI to connect to a remote and fully synced node.

Note that in order to run the node locally you will need to wait until the node is fully synced. This can take a while depending on your hardware and internet connection.

Docker image

This is a command that runs the node locally through *docker*:

```
docker run -v .:/opt/vector/ipc apexfusion/vector-node:beta-0.0.1 run
```

After running this you will have in the current working directory a file called `node.socket` which is the IPC socket that you can use to connect to the node.

Precompiled static binary

You first need to download and unpack the precompiled binaries and genesis/config files:

```
wget https://artifacts.apexfusion.org/vector-node-beta-8.9.3.0.0.0.0.
tar xvzf vector-node-beta-8.9.3.0.0.0.0.5-linux.tar.gz
```

You can connect with this command:

```
./bin/vector-node run \
    --topology share/vector_testnet/topology.json \
    --database-path ./db \
    --socket-path ./node.socket \
    --config share/vector_testnet/config.json \
    --port 7522
```

Db Sync

Vector DB Sync extracts on-chain data, storing it in a PostgreSQL database. This tool makes it easier for developers and applications to interact with blockchain data without having to process raw ledger data directly.

It works by connecting to a Vector Node through a socket and it organizes data from the node into its database relational tables.

Here you can read more about this database's tables.

Some of the tools here depend on Vector DB Sync. As of right now, you can only use vector db sync through Demeter.

Demeter has a comprehensive guide on it.

If you want to connect to it you can install psql and connect to the database with the following command:

```
# make sure to replace by the values given by Demeter
psql -h localhost -p 5432 -U postgres -d cardano
```

Here are some interesting queries you can make using psql:

```
-- Slot number of the most recent block
select slot_no from block where block_no is not null
  order by block_no desc limit 1 ;
-- Get APEX locked in scripts
select sum (value) / 1000000 as script_locked from tx_out as tx_outer
    tx_outer.address_has_script = true and
    not exists
      ( select tx_out.id from tx_out inner join tx_in
          on tx_out.tx_id = tx_in.tx_out_id and tx_out.index = tx_in.
          where tx_outer.id = tx_out.id
      );
-- Get the sum of collatereal input lost
select SUM(value)/1000000 as lost_amount
  from tx
  join tx_in on tx.id = tx_in.tx_in_id
  join tx_out on tx_in.tx_out_id = tx_out.id
    where tx.valid_contract = false ;
```

Submit API

The Submit API is a RESTful API that allows you to submit transactions to the Vector network. It is a simple and easy-to-use tool that can be used to interact with the

Vector network programmatically. The server starts with a socket connection to the Vector node and listens for incoming transactions.

```
# Fetches the precompiled binaries of the vector node repository (inc
wget https://artifacts.apexfusion.org/vector-node-beta-8.9.3.0.0.0.0.
tar xvzf vector-node-beta-8.9.3.0.0.0.0.5-linux.tar.gz

# Fetches the config file from the vector-node repository
wget https://raw.githubusercontent.com/Apex-Fusion/vector-node/refs/h
./bin/vector-submit-api \
    --testnet-magic 1127 \
    --socket-path path-to-a-fully-synced-node-socket/node.socket \
    --config ./tx-submit-mainnet-config.yaml
```

Once you've read how to create transactions you can come back and read how to submit it using javascript here:

```
// Hex encoded transaction
const cbor = "";

function hexToBinary(hex: string) {
  const length = hex.length / 2;
  const binary = new Uint8Array(length);

  for (let i = 0; i < length; i++) {
    binary[i] = parseInt(hex.substr(i * 2, 2), 16);
  }

  return binary;
}

const headers = {
  "Content-Type": "application/cbor",
};
const res = await fetch(submitURL, {
  body: hexToBinary(cbor),
```

```
  method: "POST",
  headers,
})
  .then(async (response) => {
    // Check if the request was successful (status code 200)
    if (response.ok) {
      //Parse the JSON response
      return response.json();
    }
    console.log(await response.json());
    throw new Error(`Error: ${response.status}`);
  })
  .catch((error) => {
    console.log(error);
    // Handle errors
    console.error(`An error occurred: ${error.message}`);
  });

console.log(res);
```

Vector CLI

It is a tool used to interact with the Vector network from the command line. It allow us to query the blockchain, build, and submit transactions. It is written in Haskell and is open source. You can find the source code here.

To use it you can download the precompiled binaries as shown in the Node section. You will find that the `vector-cli` binary is included in the tarball on the `bin` directory

Assuming you have a node socket pointed by the `$VECTOR_SOCKET_PATH` variable, you can query the network's protocol parameters with the following command:

```
vector-cli query protocol-parameters \
  --socket-path $VECTOR_SOCKET_PATH \
```

```
  --testnet-magic 1127 \
  --out-file protocol-params.json
```

Or you can create the signing and verification keys for an address:

```
vector-cli address key-gen \
    --verification-key-file keys/payment.vkey \
    --signing-key-file keys/payment.skey
```

Or maybe you can query the UTxOs on a given address:

```
vector-cli query utxo \
  --socket-path $VECTOR_SOCKET_PATH \
  --testnet-magic 1127 \
  --address vector_test1vq2v...
```

Or even build a transaction:

```
export ADDRESS_OUTPUT=vector_test1vq2v...
export CHANGE_ADDRESS=vector_test1vq2v...
export UTXO_1=3b0d48b0ed6507c9761011b4f9e3f1e59c1c6ce76d50b6385b15bb6
export UTXO_2=3b0d48b0ed6507c9761011b4f9e3f1e59c1c6ce76d50b6385b15bb6
vector-cli transaction build \
--testnet-magic 1127 \
    --tx-in $UTXO_1 \
    --tx-in $UTXO_2 \
    --tx-out "$ADDRESS_OUTPUT 109000000" \
    --change-address $CHANGE_ADDRESS \
    --out-file example.tx
```

Then you can sign it with this command (for this you need to get)

```
vector-cli transaction sign \
    --signing-key-file keys/payment.skey \
    --tx-file example.tx \
    --testnet-magic 1127 \
    --out-file example-signed.tx
```

And finally submit it:

```
vector-cli transaction submit \
  --socket-path $VECTOR_SOCKET_PATH \
  --testnet-magic 1127 \
  --tx-file example-signed.tx
```
